1994-03

DRUGDOG 3:0: U.S. Navy Random Urinalysis software package

Wilson, Dale E.

Monterey, California. Naval Postgraduate School

Approved for public release; distribution is unlimited.

DRUGDOG 3.0: U. S. NAVY
RANDOM URINALYSIS SOFTWARE PACKAGE

by

Dale E. Wilson
Lieutenant, United States Navy
B.S., George Mason University, 1981
B.S., Chapman University, 1990

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY from the

NAVAL POSTGRADUATE SCHOOL
March, 1994

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 15 MAR 94	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE ***DRUGDOG*** 3.0: U. S. NAVY RANDOM URINALYSIS SOFTWARE PACKAGE

5. FUNDING NUMBERS

6. AUTHOR(S) Dale E. Wilson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT *(maximum 200 words)*

Although the United States Navy has had a mandatory Random Urinalysis Program in effect for many years, there has never been a formal, standardize methodology to implement the process. OPNAV INSTRUCTION 5350.4 (series) provides guidance on what must be accomplished, but not how to accomplish it. Automation and standardization of the program through software implementation can lend confidence to personnel who undergo urinalysis testing that the program is fairly and uniformly applied to each member of the command. Informal previous attempts at developing Random Urinalysis software utilizing unstructured methods has had less than successful results. To address this problem, this thesis describes the development of a complete software application designed to automate the Random Urinalysis Program. Using previous versions of urinalysis software as templates, a standardized, structured approach to application development is used to create a new system. The Definition, Requirements, Evaluation, Design and Implementation phases of software development life-cycle are fully utilized during project development. The result is an actual working tool for the fleet. *DRUGDOG* 3.0 is a comprehensive software application that will aid individual Urinalysis Coordinators in implementing the Navy's Random Urinalysis Program within their command.

14. SUBJECT TERMS Random Urinalysis Program

15. NUMBER OF PAGES 119

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

ABSTRACT

Although the United States Navy has had a mandatory Random Urinalysis Program in effect for many years, there has never been a formal, standardize methodology to implement the process. OPNAV INSTRUCTION 5350.4 (series) provides guidance on what must be accomplished, but not how to accomplish it. Automation and standardization of the program through software implementation can lend confidence to personnel who are subject to the program that it is fairly and uniformly applied to each member of the command. Informal attempts at developing Random Urinalysis software utilizing unstructured methods has had less than successful results. To address this problem, this thesis describes the development of a complete software application designed to automate the Random Urinalysis Program. Using previous versions of urinalysis software as templates, a standardized, structured approach to application development is used to create a new software system. The Definition, Requirements, Evaluation, Design and Implementation phases of software development life-cycle are fully utilized during project development. The result is an actual working tool for the fleet. *DRUGDOG* 3.0 is a comprehensive software application that will aid individual Urinalysis Coordinators in implementing the Navy's Random Urinalysis Program within their command.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Although the United States Navy has had a mandatory Random Urinalysis Program in effect for many years, there has never been a formal, standardize methodology to implement the process. OPNAV INSTRUCTION 5350.4 (series) provides guidance on what *must* be accomplished, but not *how* to accomplish it. In a general sense, there is a certain amount of ambiguity in all Naval Instructions. This allows a military unit the flexibility to interpret a given Instruction in such a way that it meets both the intent of the Instruction *and* the needs of the command.

However, when it involves an issue as important as alcohol and drug abuse, a new perspective emerges. It can be argued that the policies and procedures regarding a program which has career-impacting potential should be of a more rigid and structured nature. Indeed, the author has witnessed numerous "spirited" discussions over the fairness of the implementation of the Random Urinalysis Program. Automation and standardization of the program could give personnel (who are subject to

the process) more confidence that it is fairly and uniformly applied to each member of the command.

There is little argument that there should be an aggressive practice in the detection and elimination of drug abusers from military service. The Navy has established a "Zero Tolerance" policy of alcohol and drug abuse. The abuse of alcohol and other drugs by Navy members can seriously damage their physical and mental health, and can jeopardize their safety and the safety of their shipmates. Perhaps most important, it can adversely affect the combat or mission-readiness of the military unit. Naval policy specifically states that:

> "...alcohol and other drug abuse is costly in terms of time lost and is a severe detriment to morale and esprit de corps. It undermines the very fiber of combat readiness, health, safety, discipline, reliability, judgment and loyalty. The abuser, as well as the abuser's shipmates and family, suffers. Alcohol and other drug abuse is incompatible with the maintenance of high standards of performance, military discipline and readiness and is destructive of Navy efforts to instill pride, promote professionalism, and enhance personal excellence" [Ref. CNO].

With these concepts as guidelines, the Navy has determined that routine urinalysis testing is the most effective means to detect and deter drug abuse. Thus, every military unit throughout the Navy must institute a mandatory Random Urinalysis Program.

B. HISTORY OF *DRUGDOG* SOFTWARE

In late 1987, this author attended a local university during off duty hours at night, studying beginning programming in pursuit of a second undergraduate degree. At that time, the idea came about to utilize these programming skills to assist his squadron in the information processing arena. The concept of developing Urinalysis software occurred during a class project that involved a random number generator utilizing the BASIC programming language.

1. *DRUGDOG* Version 1.0

The first iteration of the *DRUGDOG* software series, *DRUGDOG* version 1.0, was a primitive, simple application. It was, in fact, little more than a pure random number generator. It required the Urinalysis Coordinator to enter two numbers: the total number of personnel to be sampled, and the actual number of samples to be selected. Thus, if the Urinalysis Coordinator desired 20 selections from a roster of 300, the program would then print 20 random numbers between 1 and 300.

At this point, the Urinalysis Coordinator would have to take the command's numbered personnel roster and manually go down the

list, identifying those individuals who's number corresponded with the random number produced by the program.

Although it was felt that this method of random selection was superior to the classic "names out of the hat" or "roll of the dice" method, it still involved significant administrative manual labor. Many users of this first version of *DRUGDOG* clamored for a new product that incorporated features found in a true database system. In addition to maintaining a personnel database, they desired a more comprehensive set of printouts.

2. *DRUGDOG* Version 2.0

Early 1989 saw the second release of *DRUGDOG*, version 2.0. This application was a complete overhaul from the previous version. It had the characteristics found in most common Data Base Management Systems (DBMS), including the ability to add, delete and modify personnel records. Error-trapping was introduced during data field entry routines. The selection module automatically provided the total number of personnel in the database, and depicted the total percentage that any given sample would yield. Alphabetic listings of the selected personnel could be printed for distribution.

The user interface was carefully designed with a novice user in mind. An online User's Guide and an online Trouble-shooting Guide were particularly well received.

The cornerstone of the program was the random number generator algorithm:

- Previous Seed:= RandSeed;
- Seed:= Multiplier * (Previous Seed) * MOD (2.0EE32)

Pascal programmers will recognize most of the above pseudocode. RandSeed had been previously defined to be the number of seconds that have elapsed since midnight, January 1st of the current year. This is accomplished through an interrupt call to the DOS clock on a personal computer to obtain the current time. Therefore, even systems with the incorrect time would still provide the algorithm with the necessary information. The Multiplier was set by Borland, International, specifically for the Turbo Pascal language, version 5.0. The final component allows full 32-bit arithmetic (2 to the 32th power) for the seed to take on one full cycle.

3. *DRUGDOG* Version 2.1

This version of *DRUGDOG* repaired two system anomalies that revealed the inexperienced programming skills of the author. One scenario involved a user who had to delete a number of personnel at one sitting (as when a month had elapsed since the last Urinalysis, and a number of personnel had transferred in the interim). A logic control error did not allow more than one record to be deleted when the user entered the deletion routine. Therefore, even if the user "deleted" ten records, only the tenth record was actually deleted. This bug forced users to enter the deletion routine, delete one record, return to the Main Menu, then repeat this cycle to delete all desired records.

A second insidious anomaly involved poor error-trapping. A user could accidentally enter a "blank" in the Social Security Number field. Although this data field checked for erroneous alphabetic characters, "blanks" were not trapped. The problem was that the Social Security Number also acted as the retrieval key for modifications and deletions. Figure 1 depicts the Main Menu for *DRUGDOG* Version 2.1.

The purpose of this thesis is to use formal, structured software development methodology to design and create a complete, fully

functional software package that could be used by every command throughout the Navy to implement the Random Urinalysis program.

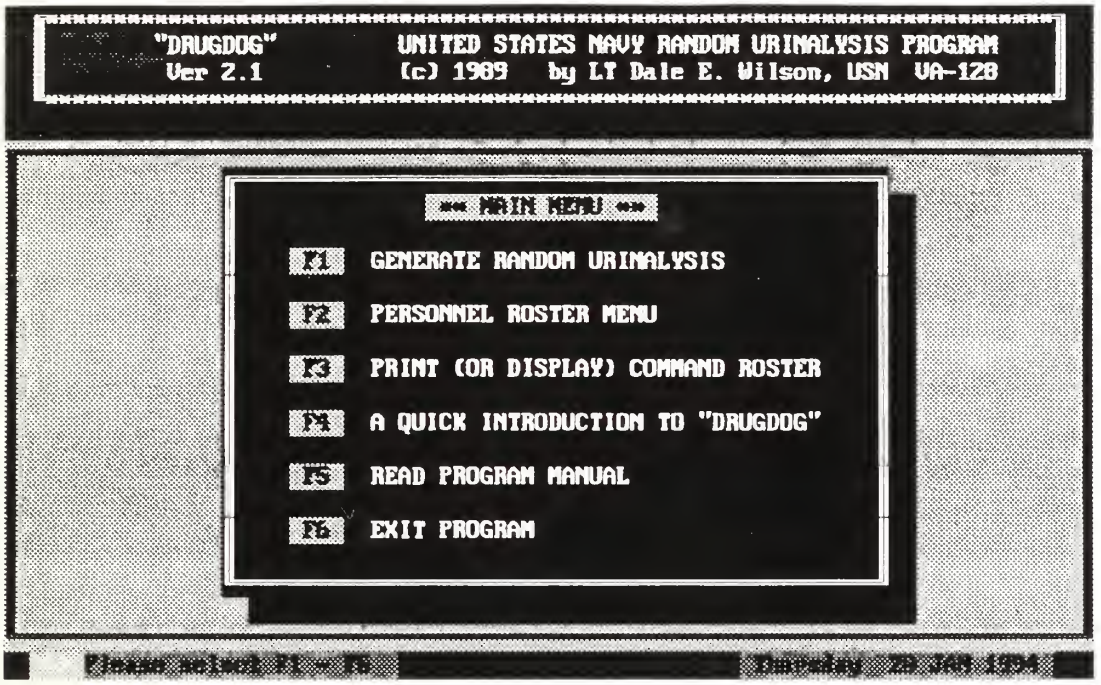


Figure 1 *DRUGDOG* 2.1 Main Menu

C. ORGANIZATION

Chapter II characterizes the general features of *DRUGDOG* version 3.0, and describes the scope of the application. Chapter III will review the database application development methodology and provide an overview of the development platform and environment. The definition, requirements, evaluation, design and implementation phases will be covered.

Chapter IV will discuss the tools and concepts behind the term "fourth-generation languages", and their impact on software development. For illustration, an overview of the 4GL environment used in the development of *DRUGDOG* version 3.0 is provided. Chapter IV concludes with a discussion of the more critical routines and modules within the code, with emphasis on the random number generator algorithms and overall documentation.

Chapter V, Conclusions, will discuss the final software package and lessons learned throughout system development. Appendices A through D include sections on requirements documentation, data dictionary, *DRUGDOG* version 3.0 application source code, and a user's guide.

II. *DRUGDOG* 3.0 SOFTWARE

Since the release of the last version of *DRUGDOG*, users from all over the world have contacted the author with suggestions to improve and enhance the software package. This chapter considers the drawbacks of previous versions of *DRUGDOG* and examines some of the user-suggested features to be incorporated in the new release.

This chapter provides a general overview of the desired features. The Definition, Requirements, Evaluation, Design and Implementation Phases are the subject of Chapter III.

A. GENERAL SOFTWARE ISSUES

1. Memory and Operating System Requirements

DRUGDOG version 3.0 will operate in the same environment as its predecessor, *DRUGDOG* version 2.1. Therefore, it must operate as advertised on IBM PC platforms (Intel 80x86 machines) utilizing MS-DOS 3.x or higher. The software package will be a standard DOS application as opposed to a Windows™ program. Not all within the user community have hardware platforms that can execute MS Windows™,

even if their command owned a copy. Hardware considerations are addressed in a later section.

Memory constraints dictate that the entire system must load and operate within conventional memory as outlined in the MS-DOS Reference Manual. After DOS and sundry device drivers are loaded, this often amounts to less than 512K of random access memory. Use of expanded or extended memory in the program is prohibited.

2. User Interface

Although *DRUGDOG* 2.1 had an interface that was cosmetically successful, it lacked standardization. For example, the main menu (previously depicted in Figure 1) utilized the function keys F1 - F6 for event handling. Other menus, however, utilized alphabetic "hot-keys" to launch a particular routine. This was not a tremendous problem, but rather was indicative of the lack of programmer training at the time.

Users want consistent color schemes and menu formats that are easy to use and provide the novice with a sense of continuity in navigating through the program. Most users desire either pull-down menus or the popular "popup" menus with transparent shadows that provide a three-dimensional effect. These menus should also have a

scrolling highlight bar to launch a particular menu item. "Hot-keys" are still very desirable as an alternative method of menu item selection.

3. Context-Sensitive Help

Lack of online, "context-sensitive" help in *DRUGDOG* version 2.1 was an issue that was often brought up by the user community. "Context-sensitive" help is a feature whereby the press of one key (usually F1) halts the application, and a series of related popup "help-fields" explain that portion of the application. Although there was a User's Guide that could be displayed on the screen from the main menu, this was a poor substitute for users who prefer pressing the standard F1 Help key at any point throughout the program.

Users asked for extensive context-sensitive help on any future release of *DRUGDOG*. They want full-page, comprehensive help on every menu, menu item, data field entry, table and listing throughout the program. If the help key is selected on a multiple-item menu, then the help screen itself should have a scrolling list of menu items that the user can select to read. These help screens should be "chained" together such that the user can go from one help screen to another without having to return to the underlying main screen.

Even with such an exhaustive and extensive system of online help, users still desire to retain the ability to have the full User's Guide available from the main menu, for either reading or printing.

4. Tables and Listings

A common complaint about early versions of *DRUGDOG* involved the manner in which tables and listing were handled. For example, users wanted the ability to scroll up or down a list of personnel, utilizing either the keyboard arrow keys or the page-up, page-down keys. They also wanted a highlight bar to "select" a personnel record for modification.

A recurring request was the ability to filter the personnel data file to display only those individuals who belong to a particular department, division or workcenter. Previous versions of *DRUGDOG* could only provide an alphabetic listing of the entire command.

A quick-find "locator" feature for searching for a record was another frequent request. This feature entails a data entry field at the top of a table or listing. The user enters the first few letters of an individual's name, and the scrolling highlight bar immediately moves to the name that matches this partial entry. For example, entering "WI" in the locator field would automatically move the highlight bar to the

"WILSON" record. This is much more convenient than entering the individual's entire social security number for record retrieval, as was required in previous versions of *DRUGDOG*.

5. Report Generation

A significant criticism of *DRUGDOG* 2.1 was the Report Generation module. Only two reports could be printed: an alphabetic listing of all personnel in the database, and an alphabetic listing of personnel selected for random urinalysis. Users felt that the inability to process printed reports to conform with local requirements was a serious drawback.

The most prevalent request was to allow listings to be printed by Division or Workcenter. For large commands that conduct many random selections per sampling period, the ability to print and distribute Urinalysis listings by Divisions or Workcenter would eliminate a heavy administrative burden on the Urinalysis Coordinator. Ideally, future releases might also generate preprinted labels to be affixed to the urine sample containers.

6. Program Installation

Finally, there were numerous requests to include a program to install the system on a hard drive. Earlier versions of *DRUGDOG* either

had a simple batch file for hard drive installation, or none at all. This was justified (at the time) due to Privacy Act considerations, and also due to the very nature of the application. It was felt that it was more secure to execute the program from a floppy drive, where the disk could be removed and safeguarded from possible malicious tampering.

Many stand-alone personal computers, however, now incorporate good ADP Security features that include multi-layer access control. Therefore, an installation module would be an appropriate addition to the next release of *DRUGDOG*. This module should have the ability to install the application from either drive A or B, and have meaningful error-trapping to ensure proper and successful system installation.

III. APPLICATION DEVELOPMENT

The five development phases used in the production of *DRUGDOG* version 3.0 will be discussed in this chapter. The methodology of each phase will be discussed followed by how that phase was applied in the development process. Throughout the development cycle, significant consideration was placed on the anticipated user community, with particular emphasis on the following aspects of the user interface: [Ref. SOMMERVILLE]

- The interface should use terms and concepts which are familiar to the anticipated class of user.
- The interface should be appropriately consistent.
- The user should not be surprised by the system's characteristics.
- The interface should include some mechanism which allows users to recover from their errors.
- The interface should incorporate an extensive system of user guidance and context-sensitive help.

These objectives keep the focus on the most important element when producing any software application: the users. End user needs must be the overriding consideration throughout system development. This cannot

be overemphasized. If an Information System does not satisfy the end user, than the intended objectives can not be met.

DRUGDOG version 3.0 fulfills these objectives. It was designed to assist commands in managing their Random Urinalysis Program. It provides the Command Urinalysis Coordinator a vehicle that can automatically select a truly random list of personnel to undergo testing.

A. PHASE I: DEFINITION PHASE

1. Methodology

The purpose of the Definition Phase includes preliminary activities with the major goal of determining what needs to be accomplished. The development team must be formed, the scope of the project established, and feasibility (in terms of cost, technical requirements and time constraints) assessed.

2. Application

The goal of this thesis is to provide a fully functional system application for automating and standardizing the Navy's Random Urinalysis Program as outlined in OPNAV INSTRUCTION 5350.4 (series). Since it is an enhancement to an existing system, it was decided that the scope of this work warranted development as an individual

thesis project. All feasibility issues were met satisfactorily. Development would be performed on a 80386, 25 Mhz IBM-compatible PC owned by the thesis student. This resulted in negligible cost. The availability of various application development software owned by both the Naval Postgraduate School and the thesis student eliminated another cost. A time span of nine months with commencement in June, 1993 and system completion by February, 1994 was considered feasible. With the Definition Phase completed, the next step was to begin the Requirements Phase.

B. PHASE II: REQUIREMENTS PHASE

1. Methodology

Identifying the objectives of the proposed system in detail is the goal of the requirements phase. Requirements are the blueprint that will be used to design and implement the new system. Before being able to move on to development, the developer must know exactly what the system is supposed to do. It is not only important that the system be built correctly, but that the right system be built. Proper definition of the requirements can prevent future maintenance trauma.

There are two major tasks in defining database requirements. The first is to identify the objects. Objects are a collection of properties which depict an item to be implemented in the database [Ref. Kronke] These objects are most effectively identified by examining previous versions of the application, and conducting a series of interviews with the expected users. After initial interviews, a prototype may be built and demonstrated to receive further user design input.

2. Application

Interviews commenced in June of 1993 with two perspective beta-testers, who were selected based upon their substantial previous experience with the Random Urinalysis Program. Interviews were also conducted with the current Urinalysis Coordinator at the Naval Postgraduate School.

The initial interviews lasted approximately three weeks. Working with initial data requirements, a prototype *DRUGDOG* 3.0 application with sample input screen and reports was presented to the users for review. Several suggestions and requests were presented and discussed during the review. This cycle repeated itself several times over the next three months. Fortunately, the selected development environment contained a fourth-generation language (4GL) that was well

suited for recurring prototyping. The ample time schedule also allowed for these constant changes.

a. Data Requirements

Prior to this discussion, it is appropriate to provide definitions for terms used in describing the data requirements [Ref. Kronke]. An *object* can be described as the constructs and conventions used to create a model of the users' data. An *object property* is a property (characteristic) of an object that is, itself, an object. An *object definition* is a named collection of properties that sufficiently describes a distinct identity. Finally, the *domain* of a property is the set of all possible values the property can have.

The PERSONNEL object, as determined through the interview/prototyping process, is shown in the Object Diagram, Table [1], Appendix A. This is the single most important object in the system. The central focus of this application involves randomly selecting these "objects" for urinalysis.

The Department property is an *object property*, which means that this entity characteristic is actually another object. The same is true for the DIVISION and WORKCENTER objects. Additional PERSONNEL object data information is supplied in Appendix A. Table

[2] provides the object definition which lists all of the object properties and each property domain. Table [3] is the Domain definition which specifies formats of each domain. Tables [4] and [5] depict the Display, Update and Control Mechanisms. This information is used for the database design in Phase IV.

b. Functional Requirements

Functions required by the *DRUGDOG* 3.0 software were patterned after the previous version of *DRUGDOG*, version 2.1. These functions include record entry, display, editing, deletion, and report generation. Additionally, OPNAV INSTRUCTION 5350.4B was meticulously scrutinized to ensure program compliance with this governing directive.

The data flow diagram (DFD), Figure 2, depicts a graphic model of the *DRUGDOG* system to be used as an aid in design. The DFD is comprised of four elements: the *data flow*, represented by an arrow, indicates data movement between a source and sink;the *process*, represented by a circle, indicates data manipulation; the *data store*, represented by an open ended rectangle, is data stored before or after processing; and the *source/sink*, represented by a closed box, either provides or receives the data.

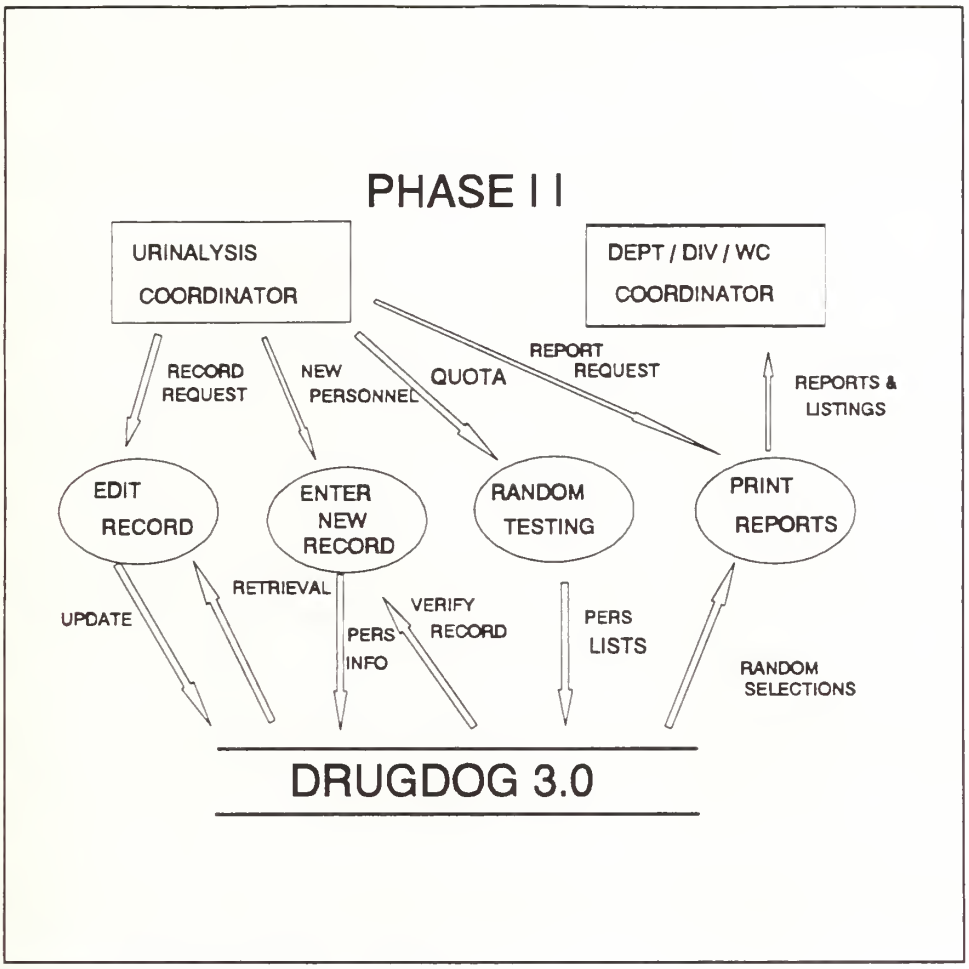


Figure 2 Data Flow Diagram

c. Hardware Considerations

(1) Target Platform Because the predominate system currently in fleet-wide use is the Zenith Z-248, it will be considered the minimum platform for the *DRUGDOG* software package. The vast majority of the Zenith Z-248 computers were distributed with a single 360K 5 1/2" floppy drive, and a 20 Megabyte hard drive. Therefore, to execute the program from floppy disk only (due to ADP security issues),

a minimum requirement is that the entire *DRUGDOG* application, including the DBMS, data, index, manual and help files, and the installation program, be no larger than 360 Kilobytes in total size.

(2) Alternate Platforms Testing of the new version of *DRUGDOG* will also be conducted on hardware that includes the Unisys 80386 systems procured under the Desktop III Contract. Experience has shown that an application that behaves as advertised on an Intel 80286 PC with an EGA monitor (such as a Zenith Z-248), generally behaves as advertised on higher level machines such as those procured under Desktop III (80386SX systems equipped with VGA graphics). Hardware is **not**, however, the key issue in this generalization. The software operating environment is often the culprit in what might otherwise appear to be system incompatibility. Different versions of operating systems, various memory managers and memory-resident programs are all a potential source of unexpected results. In the next section, we discuss Phase III of the development process, the Evaluation Phase.

C. PHASE III: EVALUATION PHASE

1. Methodology

Using the information gathered during the Requirements Phase, this development stage typically consists of an evaluation of several items of concern to both the developer and the customer. During evaluation phase, the application development architecture is chosen, user requirements reexamined, and feasibility reassessed. The evaluation phase allows the end user and system designer to meet again to ensure user needs are satisfied. Detecting design flaws or revising requirements needs to be identified in the early stages. Significant savings in time and money will be realized if corrections are made before the implementation phase.

2. Application

The software selected for developing the application was the Clarion Professional Developer, version 2.1. This system is designed for developing high-quality business application programs for microcomputers. In evaluating prospective application development platforms, Clarion was selected due to its 4GL programming strengths. Some of these strengths are described below.

In Clarion, support for screens, reports and files is built-in, not supplied as a toolbox. It allows the design of versatile screen structures. A screen layout can be a window. When a window is opened, it "pops-up" on the video monitor; when it is closed, it disappears. A help window can be tied to a field, making context-sensitive help windows a natural part of the development process.

The real benefit of screen declarations is provided by the screen formatting capability of the 4GL's Editor. Editor's Screen Formatter copies a screen structure, displays the screen layout, accepts changes from the keyboard/monitor, and replaces the screen structure in the source program. Screens can be created and maintained indefinitely, thereby supporting code reusability.

User requirements were refined through interviews with the two perspective beta-testers, and the current Urinalysis Coordinator at the Naval Postgraduate School. Discussions were also held with Professors Schneidewind and Short, co-advisors of this thesis. A thorough examination of user needs was completed before advancing to the next phase.

A complete review of the evaluation phase minimizes user dissatisfaction and reduces the number of changes that must be made

after the system is built. The outcome of the evaluation phase resulted in a set of finalized user requirements that would be technically feasible using the selected development platform. The data model was refined to reflect these changes. With this improved set of definitive requirements, the evaluation phase was completed. The Design Phase is discussed in the next section.

D. PHASE IV: DESIGN PHASE

1. Logical Database Design

In the design phase the logical model is translated into specific data structures and relationships. The semantic object model was revised to reflect these data structures. Each property of an object is an attribute to the relation. Logical database design is generic; specific design requirements for programming in the Clarion 4GL will be covered in the Physical Database Design section. The requirements determine what is wanted from the system and the design determines how to accomplish those goals. Logical design plans developed from the object diagrams and object definitions consists of relation diagrams, relation definitions and the constraints on the relations.

The PERSONNEL object was transformed into the PERSONNEL relation as depicted in Figure 3.

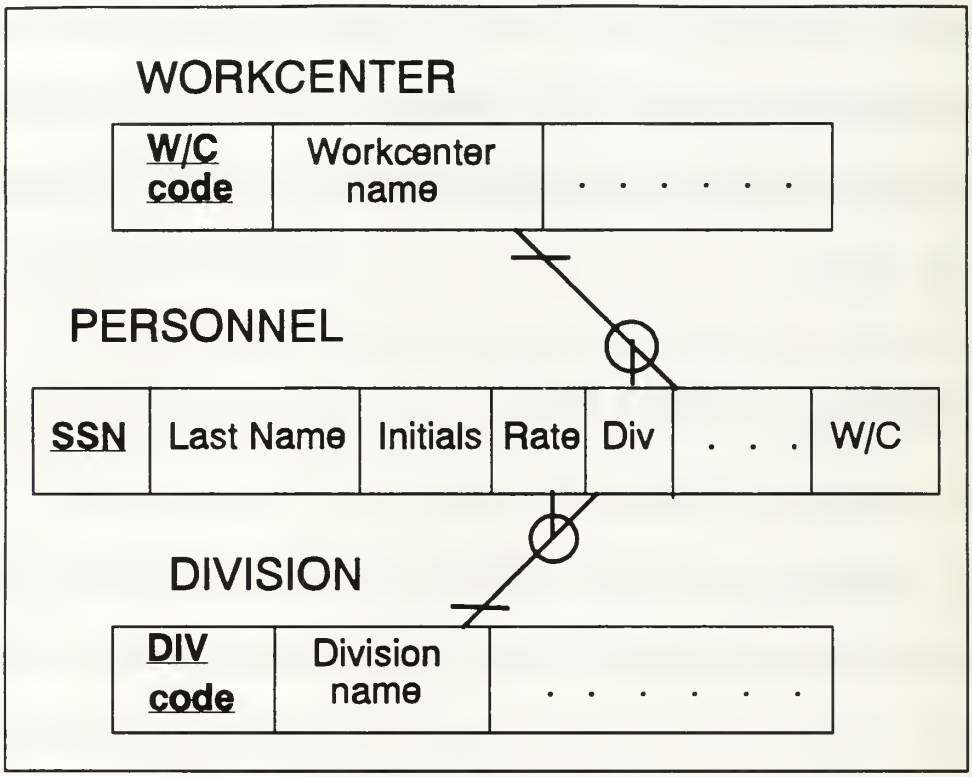


Figure 3 Personnel Relation Diagram

The PERSONNEL relation uses an individual's social security number as its primary key. (This is **not**, however, the key used in the r: om selection process, which is discussed in Chapter IV.) A key is an attribute that allows the non-key attributes to be entered and retrieved from the object. The PERSONNEL relation is related to the WORKCENTER relation in a one-to-many binary relationship. The "fork" at the WORKCENTER end of the relationship line means that there are

potentially many personnel for each workcenter within the WORKCENTER object. The absence of a fork at the other end indicates that each personnel can be assigned to at the most, one workcenter at any one time. The circle on the line means that the relationship from PERSONNEL to WORKCENTER is optional. A WORKCENTER does not have to have any personnel assigned to it. (This is to create a "dummy" workcenter for the random generator algorithm, as discussed in Chapter IV.) The bar at the other end indicates that a PERSONNEL record must correspond to a WORKCENTER record. DIVISION is linked to PERSONNEL in much the same manner.

The relational database model is based on the concept that the data is stored in two-dimensional tables referred to as relations. Each row in the table represents a record. Each column represents a field. The entire table (relation) is what is known roughly as a file. *Normalization* can be defined as the process of evaluating a relation to determine if it is in a specified normal form, and if necessary, converting it into relations that are in that specific normal form [Ref. 1]. This normalization process is handled internally by the 4GL compiler. It ensures no insertion, deletion or modification anomalies can exist, leaving all relations in the (DK/NF) Domain Key/Normal Form.

2. Physical Database Design

This stage of the design phase will transform the logical database into a physical design of the specific data elements that are required for programming the application with the Clarion 4GL. Each field must also be categorized as one of eight general data types allowed by Clarion:

1. BYTE - one-byte unsigned integer, 0 to 255.

2. SHORT - two-byte signed integer variable in Intel 8086 word integer format. Range is from -32,768 to 32,767.

3. LONG - four-byte signed integer variable in Intel 8086 short integer format. Negative numbers are represented in standard two's complement notation.

4. REAL - eight-byte floating point variable in Intel 8087 long real (double precision) format.

5. DECIMAL - variable length packed reals, each byte holding two decimal digits.

6. STRING - fixed-length character string up to 255 characters long.

7. GROUP - construct that allows multiple consecutive variables to be reformed as a group by a single variable name. GROUPS help organize complicated programs by keeping related data together.

8. EXTERNAL - specifies a parameter of a procedure or function that is "passed by address". Attributes of EXTERNAL permit arrays, screens, reports, files, keys, and tables to be passed as parameters.

With the exception of DECIMAL, all of the above data types were utilized in the development of *DRUGDOG* version 3.0. This 4GL has a rich set of data types. A language rich in data types, however, ought to be intelligent in dealing with them. A mixed expression should not need a conversion function to multiply a BYTE by a SHORT producing a LONG. The compiler knows the data types; that is why they were declared. Therefore, a variable of any data type can be moved or added to a variable of any other data type without conversion functions or cast directives. Table [6] in the Data Dictionary (Appendix B) lists all *DRUGDOG* data file elements in proper format.

3. *DRUGDOG* 3.0 Application Design

The implementation of this application is the collection of menus, forms, reports, and programs that perform the functions of the system required by the users. Before proceeding to the Implementation Phase the final task is to design the application. Once the basic designs for *DRUGDOG* 3.0 (Appendices A and B) were determined, a quick prototype was developed to demonstrate the menus, input form screen, and reports to the users. User-requested modifications to the prototype were incorporated to form the final application design.

a. Menu Design

Since the development of *DRUGDOG* version 3.0 was an enhancement (albeit a substantial overhaul) to an existing system already in use, it was decided to follow the current structure in designing *DRUGDOG* 3.0 menus. This would allow easier program integration in the implementation phase and ease the user transition to the new system. The menu hierarchy design is illustrated in Figure 4.

The main menu features six main options, four which have their own "pop-up" menus for option selection. The menus feature a scrolling highlight bar which determines option selection. Alternatively, the first letter of any menu item serves as a "hot-key" to launch that item. These sub-menus maintain the same "look and feel" as the main menu, which gives users a sense of continuity in navigating through the menu hierarchy.

The standard F1 "help" key can be pressed at any menu for a complete explanation of menu options. All pop-up menus have translucent shadows that lend a graphical three-dimensional effect to the interface.

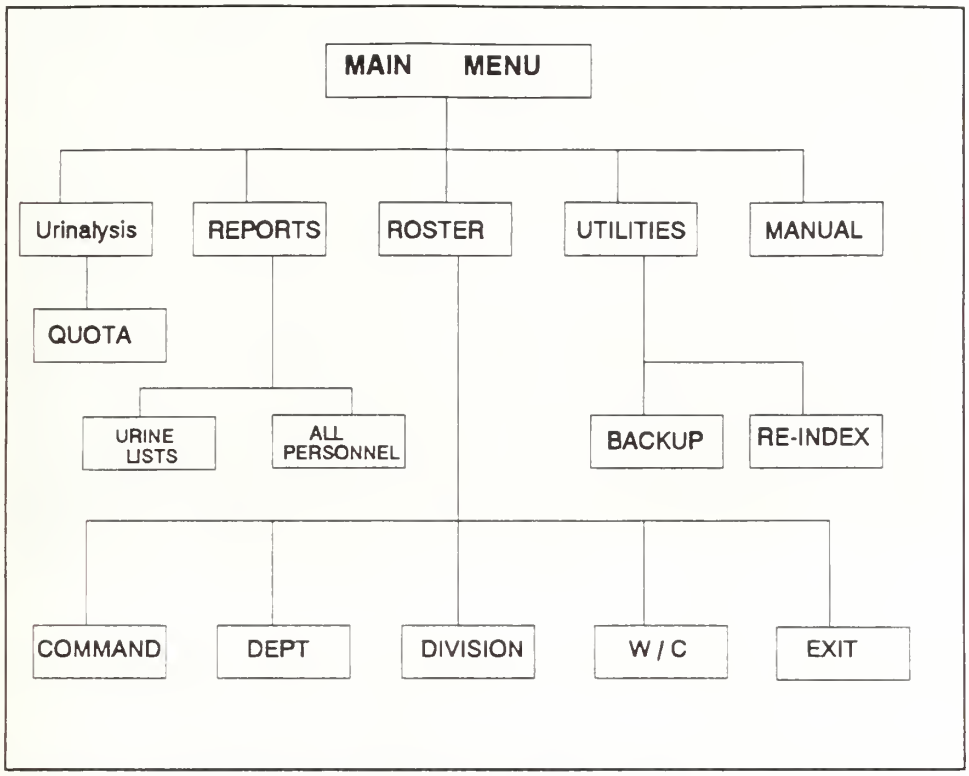


Figure 4. *DRUGDOG* Menu Hierarchy

b. Screen and Table Design

Variations in the screen layout design can either ease or hinder system use. Screen design begins with determination of the information and fields that will be placed on the screen and then effectively designing the arrangement so that the data will fit within the screen limitations.

A unique design for the record entry form was developed as a result of user requests during prototyping. The Roster Maintenance

option from the main menu allows a scrolling table of personnel to be displayed. This table can be "filtered" such that only personnel from a particular workcenter, division or department are displayed. A highlight bar can be positioned on an individual record. If the user then presses <RTN>, the record edit form automatically pops up, allowing instant record modification to occur. After the record is modified, data from linked relations are automatically updated as well.

Another characteristic of a good interface is the use of varying colors as a means to convey information. Input forms were designed utilizing an eye-catching bright white on red background. Each individual input field, however, was highlighted utilizing a bright yellow on black color scheme. This alerts the user to which field is currently being edited. As with the menus, all input forms are popup windows with translucent shadows. Figure 5 illustrates a typical record modification interface.

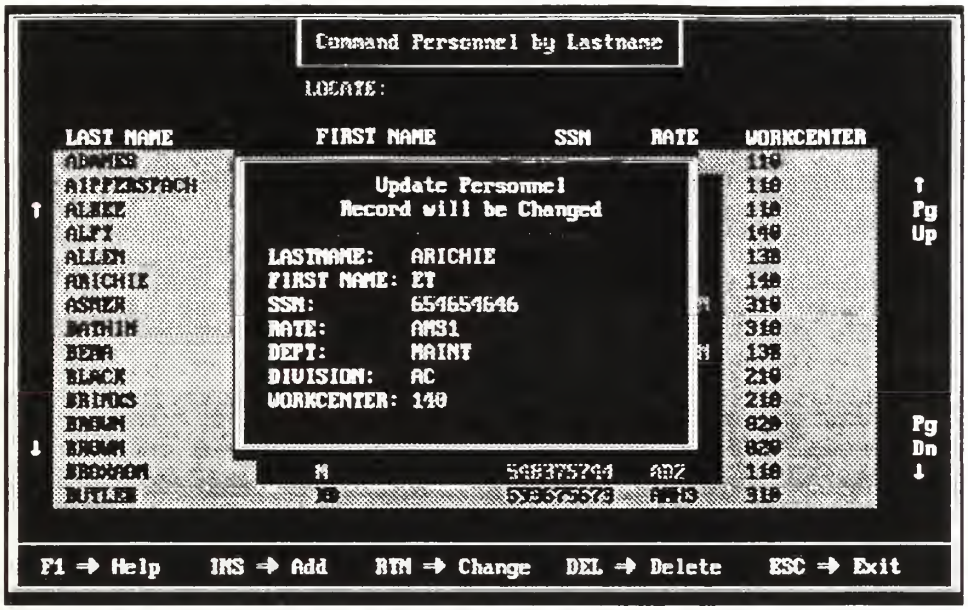


Figure 5 *DRUGDOG* Input Form

c. Report Generation

The earlier versions of *DRUGDOG* allowed for only two types of printed reports: an alphabetical listing of the entire command, and an alphabetical printout of a given random urinalysis sampling. Users requested the ability to print out both the entire roster or any given random selection using any one of three filters: by Department, by Division, or by Workcenter. This allowed large commands with sizable quotas to print and distribute random urinalysis listings in a compartmentalized manner, eliminating wasteful redundancy in paper cost and wear and tear on the command's copying machine. Finally, the

last development phase , the Implementation Phase, is discussed in the next section.

E. PHASE V: IMPLEMENTATION PHASE

1. System Programming

Constructing the system in accordance with the design is the fundamental task of implementation. Breaking large application programs into smaller source modules usually creates problems. First, every label used but not declared in a module must be designated as a global where it is declared, and as an external everywhere it is used. Second, when a module is compiled, it must be linked with the other modules before it can be tested. Third, link errors often sends one searching through modules looking for unresolved global references.

To solve these problems, an application consists of a program module and optional member modules. All global data is declared in the program module. If a member module has global references, it names the program where they are declared, not the label of every label used. For a member module, the compiler reads the program module's symbol table and "compiles in" the global addresses, allowing testing without linking.

A program containing member modules, procedures or functions must have a map structure naming them. When a program module is changed, the member modules in the map structure are "stream compiled". A stream compilation compiles the program module, then compiles each member module to correct its global references.

The 4GL's Processor utility program is used to test and debug programs. It loads the compiled modules and executes them. The interactive debugger allows one to view and change variables, set break-points, trace, single-cycle, and jump around a program, all without recompiling. When testing is completed, the 4GLs' Translator utility program produces standard object modules from the compiled modules. The object modules are linked with the 4GL's library to produce a standard .EXE program that can be executed from DOS.

2. Testing

Each section of the system was thoroughly tested for correct function using a black box testing procedure. Black box testing is a testing method where inputs are provided to the system with subsequent examination of the outputs to ensure the systems overall function is as advertised. Black box testing attempts to find errors in the following categories: [Ref. PRESSMAN]

1. Incorrect or missing functions and interface errors
2. Errors in data structures or external database access
3. Performance errors
4. Initialization and termination errors

This testing method does not concern itself with internal functions, rather it checks the correctness of the system as a whole. This is particularly well suited for an application developed with a 4GL such as Clarion. The vast majority of the code is generated using DESIGNER, the main module of the Clarion developer platform. DESIGNER takes developer menus, tables, reports, files, keys, and constructs an application template. It then generates (syntax) error-free source code, including comments. Relations are extensively tested and put in the Domain Key / Normal Form to ensure there are no addition, deletion or modification anomalies. Entry fields are automatically error-trapped to only allow correct alpha-numeric input.

Approximately 140 fictitious test records were entered into the system with no discrepancies. Over 100 random selection runs were performed. There were no discernable patterns in the random selections. A group test (comprised of the "beta-testers, developer, and prospective

users) was the last stage to see if there were any final functional problems that the programmer may have overlooked. The group agreed that the system appeared to work as advertised.

3. Installation

The last stage of the Implementation Phase is installation. There are four primary methods to install a new system. Two are the "pilot" and "phase-in" methods, neither of which apply here. A third is to completely abandon the old system and simply use the new one. The fourth method is to run parallel systems until the new system is fully integrated into the organization. The latter is often the preferred method until users have learned the new system and have confidence that it is reliable.

Distribution of the system will be similar to the previous version of *DRUGDOG*. The entire package will be made available to the Department of Defense via the Naval Computer and Telecommunication Stations' (NCTS) electronic Bulletin Boards (BBS). It will be in a compressed format utilizing PKZIP compression utilities. Once a command downloads and "unzips" the *DRUGDOG* package, the installation program *DDINSTAL.EXE* (which accompanies the application) will install the complete system from any floppy drive onto

drive. The sample roster that is included will allow the user to immediately experiment with the application.

The following chapter examines the impact that fourth-generation languages (4GL) have had on software development. To illustrate these concepts, an overview is provided of the 4GL used in the development of *DRUGDOG* 3.0.

IV. DEVELOPMENT ENVIRONMENT

This chapter examines some of the more interesting aspects that fourth-generation languages (4GL) have brought to the software development arena. There has been much press in recent years on how 4GLs have helped "revolutionize" software development. It can be argued that there are as many definitions of a 4GL as there are people to define them. Yet, there are certain concepts or features that are usually common. To illustrate these concepts, the Clarion Integrated Development Environment (IDE) will serve as an example 4GL, with emphasis on the main development modules that comprise the IDE.

The chapter concludes with a brief discussion of the coding of algorithms that are at the heart of *DRUGDOG* version 3.0. As with many fourth-generation languages, Clarion generates much of its own source code. However, several of the more critical modules had to be "hand-coded". These include the logical and physical random selection process, and mechanisms that prevent duplicate random selections.

A. FOURTH GENERATION LANGUAGES

The term "fourth-generation language" (4GL) encompasses a broad array of software tools that have one thing in common: each enables the software developer to specify some characteristic of software at a high level. The tool then automatically generates source code based on the developer's specification. Since the early 1980s, the software industry has used the term 4GL to describe varying types of productivity tools.

Three classes of 4GLs have emerged in recent years: end-user tools, application developer facilities, and tools that emerged from relational DBMS products. However, the boundaries between these classes are growing fuzzier. These languages offer major productivity improvements over third-generation languages (3GLs) such as Cobol, Fortran or PL/1. Faster hardware, alternative platforms, distributed architecture and improved software performance have eliminated many earlier efficiency concerns about the performance of fourth-generation languages for large, complex systems [Ref. Weitz].

The major characteristic of a 4GL is that the tool should enable users to develop applications at a rate several times faster than a 3GL would for that same application. These sets of tools include some or all of the following [Ref. Rinehart, Weitz]:

- nonprocedural languages for database query
- report generation
- data manipulation
- screen interacter and definition
- code generation and compiling
- high-level graphics capability
- integrated development platform

Most 4GLs feature a nonprocedural language. This means that the user specifies what to do, but not the details of how to do it. The language is also at a higher level of specification than 3GLs, which translates into significantly less code. [Ref. Rinehart]

4GLs must offer facilities for complete application development, rather than addressing only a piece of it. Many products are available today for building graphical front ends to applications that run in client/server environments. However, since these tools only address the front end or user interface, they are not considered the full-function 4GLs such as the one used for development of *DRUGDOG*.

Prototyping methodologies became popular in the early 1980s concurrent with the acceptance of 4GLs [Ref. Weitz]. Prototyping and

4GL development work well together. The 4GL code is easy to modify and refine until it satisfies the users' requirements and becomes the production system. Throughout the development of *DRUGDOG* 3.0, the 4GL's rapid prototyping capability proved the critical feature that allowed application completion within the given deadline.

To illustrate the above discussion, the following section describes how Clarion, the software selected for the development of *DRUGDOG*, fits within this 4GL category through a examination of its development environment. Many of the issues and concepts presented above are found within the description of each utility tool.

B. INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

Most 4GLs provide some variant of an integrated, interactive program development environment. To the programmer, a seamless, full-function integrated development environment is considered a critical feature of a 4GL. This environment provides utility programs that greatly simplify the common tasks one does as a programmer, such as creating custom screens, help windows, data files, and reports. Each utility program also features context-sensitive help windows that provide reference information and facilitate error correction. Figure 6 depicts the Main Menu of the Clarion IDE. While running under this environment,

Clarion keeps track of the source code files that are currently in work and passes them to each utility.

Figure 6 Clarion Main Menu

1. *DESIGNER* Utility

The *Designer* utility is the heart of the Clarion IDE. It can be considered the main application development and prototyping tool. It produces source code that is free of syntax errors, based on the files, screens and reports that a programmer designs. This module also produces all the "connecting logic" that actually makes the program complete. In fact, depending on how sophisticated an application is, one

may be able to create the entire application using only *Designer*. Typically, the utility is used in one of three ways:

- It can be used to create fully functional application programs. It can be used to create a wide range of business oriented applications, from simple mailing list to a hard-core order entry/inventory/invoicing program.

- It can be used to prototype an application or as a "first step" in application development. It can develop the template or "shell' of a program, then the Editor can be used to hand-code any unique features desired. This was the method employed in the development of *DRUGDOG* 3.0.

- It can be used to create a program "frame" that includes special hooks to pull in existing Clarion, C/C++, or Assembly language procedures.

Designer's Application Summary window provides an instant picture of the current state of the design of an application. It illustrates the relationship of the various procedures of the application and their current status. Figure 7 depicts part of *DRUGDOG* 3.0's Application Summary window.

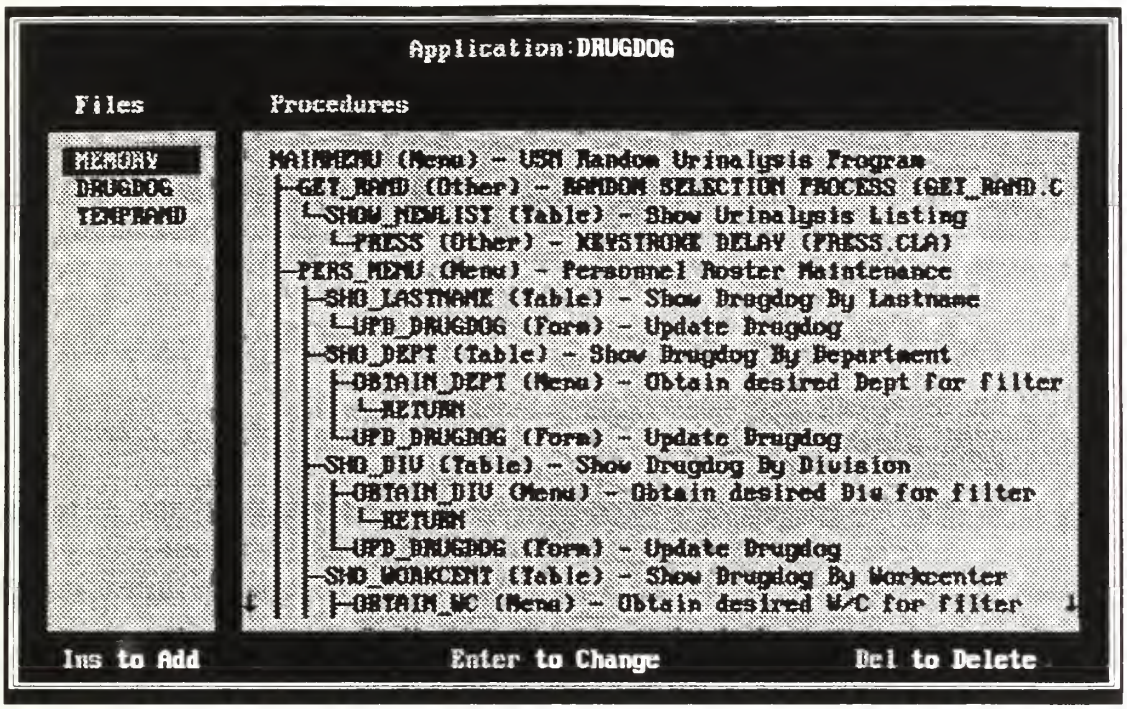


Figure 7 *DRUGDOG* 3.0 Application Summary Window

2. *EDITOR* Utility

A 4GL's editor is used to "hand write" source code that can not be created using the code generator described above. The Clarion *Editor* has three parts: a Source Editor, a Screen Formatter, and a Report Formatter.

- Source Editor is used to write and edit Clarion source code, to modify the source code generated by the utility, and to locate and correct errors identified by the 4GL's compiler.

- Screen Formatter is used to create and format screens and to generate the source code for screen structures. It can also modify

existing screen structures created by the Screen Formatter or with the utility.

- Report Formatter is used to create and format printed reports the application performs and to generate the source code for these reports. It can also modify existing report structures created by either the Report Formatter or Designer.

The *Editor* is a full-screen, ASCII line editor. In addition to traditional editing functions, it also has a number of labor-saving features. For example, it can define keyboard macros, set automatic indentation, restore deletions, and edit text in one column without affecting the position of adjacent columns. It can also edit two files simultaneously, allowing the exchange of data between the active file and the inactive file.

3. *COMPILER* Utility

The *Compiler* utility reads the Clarion source code and translates it into compact, high-level pseudo-code. This pseudo-code, in turn, is read and executed by the *Processor*. As the *Compiler* reads the source code, it detects and lists any errors. The errors can then be displayed in the *Editor*, enabling one to make corrections before processing. The *Compiler* offers a fast and efficient way to compile all of the modules of a program in a single pass. Stream compiling, which is necessary only when global variables have been changed, proceeds

quickly because the *Compiler* and the global symbol table need to be loaded only once. The *Compiler* produces an optional program listing with page titles, subtitles, and line numbers. The listing also displays the depth of the nested logic within the program.

4. *PROCESSOR* Utility

The *Processor* utility is used to load and execute the processor files generated by the *Compiler*. It generates a processor file for each PROGRAM module compiled, as well as any MEMBER modules belonging to that PROGRAM. The processor can than "run" the program as if it were a regular .EXE file.

During program execution, the programmer can access an interactive *Debugger* by pressing Ctrl-Break at any time. This *Debugger* allows the programmer to view and change the current value of up to six program variables simultaneously. The *Debugger* allows the programmer to trace the progression of the programs logic, to specify break-points, and to step through a program at the line-by-line source code level. This utility was employed extensively during the development of *DRUGDOG* 3.0.

5. *TRANSLATOR* Utility

The *Translator* utility is usually the last step in the creation of a standalone executable program. Typically, the program has been tested and debugged prior to translation. The *Translator* produces relocatable Intel 8086 native-code object modules from the processor (.PRO) and symbol (.SYM) files of the compiled program. These translated object modules, which have the extension .OBJ, can then be linked with the 4GL's Run-Time Library to produce an executable program with an .EXE extension. Although the .OBJ modules can be linked with any linker supplied by DOS 3.3 or higher, Clarion also includes the high-quality grade linker *PLINK86plus*, which has the advantage of supporting overlays for large programs.

C. RANDOM SELECTION CODING

As discussed above, there are often times that special functions or procedures required for a given program can not be accomplished using the 4GL's code-generator alone. These modules must be "hand-coded" using the *Editor*. Such was the case in the development of *DRUGDOG* 3.0 when it came to the actual Random Urinalysis Selection module. Fortunately, the Clarion language has a built-in function under the Mathematical Functions section called RANDOM(*low,high*). The

RANDOM() function generates a random integer between the *low* and *high* parameters inclusively (therefore including the endpoints). The *low* and *high* points can be any numeric expression, but only their integer portion is used to form the inclusive range. The return value is a long integer, which turned out to be exactly the data type needed as the algorithm progressed.

1. Random Number List Generation

As previously mentioned, the random function requires two parameters. For its employment in *DRUGDOG*, the *low* parameter would always be 1 (obviously there will never be a need to select zero or less personnel for urinalysis testing). The *high* parameter will always be the total number of personnel in the database at the time the urinalysis test is conducted. Clarion provides an internal function called RECORDS() that returns the total number of records (of personnel) in the database in a variable called TOTAL_RECS. Appendix C lists the resulting source code. Basically, this module can be described as follows:

A simple loop from one to QUOTA_NUMBER (which was previously obtained from the Urinalysis Coordinator) fills an array of long integers called QUOTA_HOLDER[]. A "nested" loop then compares each newly selected random number with all previous random numbers

in the array. If a duplicate number is detected, then the duplicated number is deleted, the loop counter is reset to compensate for the loss, and the outside loop continues as before. This cycle is repeated until the entire array is filled with non-repeating, randomly selected numbers.

2. Physical Record Selection

Now that an array of random numbers is available, the actual selection of personnel is described as follows: a "GET" statement uses the individual random number as a record "pointer" to pull the record that physically resides at the "address" provided by the pointer. That record is then copied into a temporary record holder (RAN:RECORD) and is added to a new (Urinalysis) data file called RANDFILE. This cycle is repeated until the array of random numbers is exhausted.

This manner of random selection is independent of any external influence other than the desired QUOTA number that the Urinalysis Coordinator provides. Selections are based on the physical location of the records, which are constantly being changed in the normal course of record deletions, additions, and file re-indexing.

V. CONCLUSIONS

The purpose of this thesis was to provide a complete, fully functional software package that could be used by every command throughout the Navy. Although the Navy's mandatory Random Urinalysis Program has been in effect for over a decade, there has never been a formal, standardize methodology to implement the process. OPNAV INSTRUCTION 5350.4 (series) provides guidance on what *must* be accomplished, but not *how* to accomplish it. The automation and standardization of the process through software lends confidence to personnel who are subject to the program that it is fairly and uniformly applied to each member of the command.

Since the author developed previous versions of the Random Urinalysis application known as *DRUGDOG*, commands from all over the world who have acquired and implemented the software have called for an updated version to incorporate various features and enhance flexibility. These features were discussed in Chapter II. The reason for such a substantial number of requests was due largely to the lack of

formal, structured approach to software development in these early versions.

To implement user requests, the author utilized the formal software development techniques as taught throughout the Information Technology Management curriculum at the Naval Postgraduate School. The five main development phases include the Definition, Requirements, Evaluation, Design and Implementation Phases. These phases were examined in great detail in Chapter III.

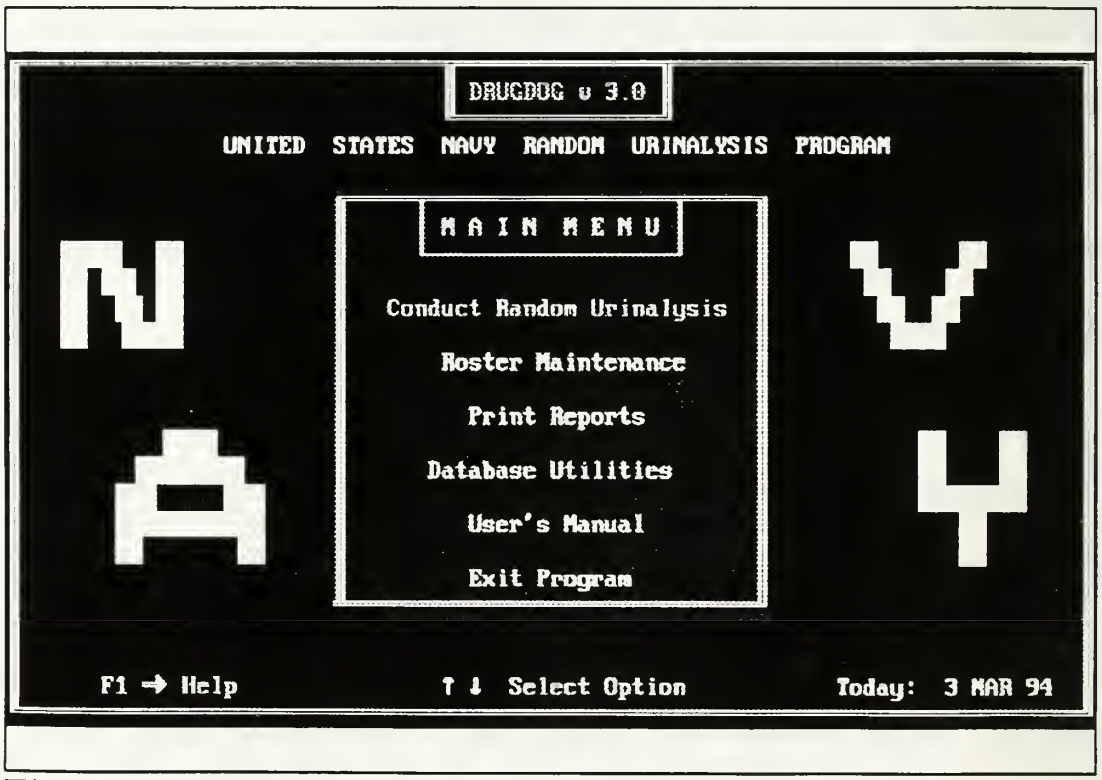


Figure 8 *DRUGDOG* version 3.0 Main Menu

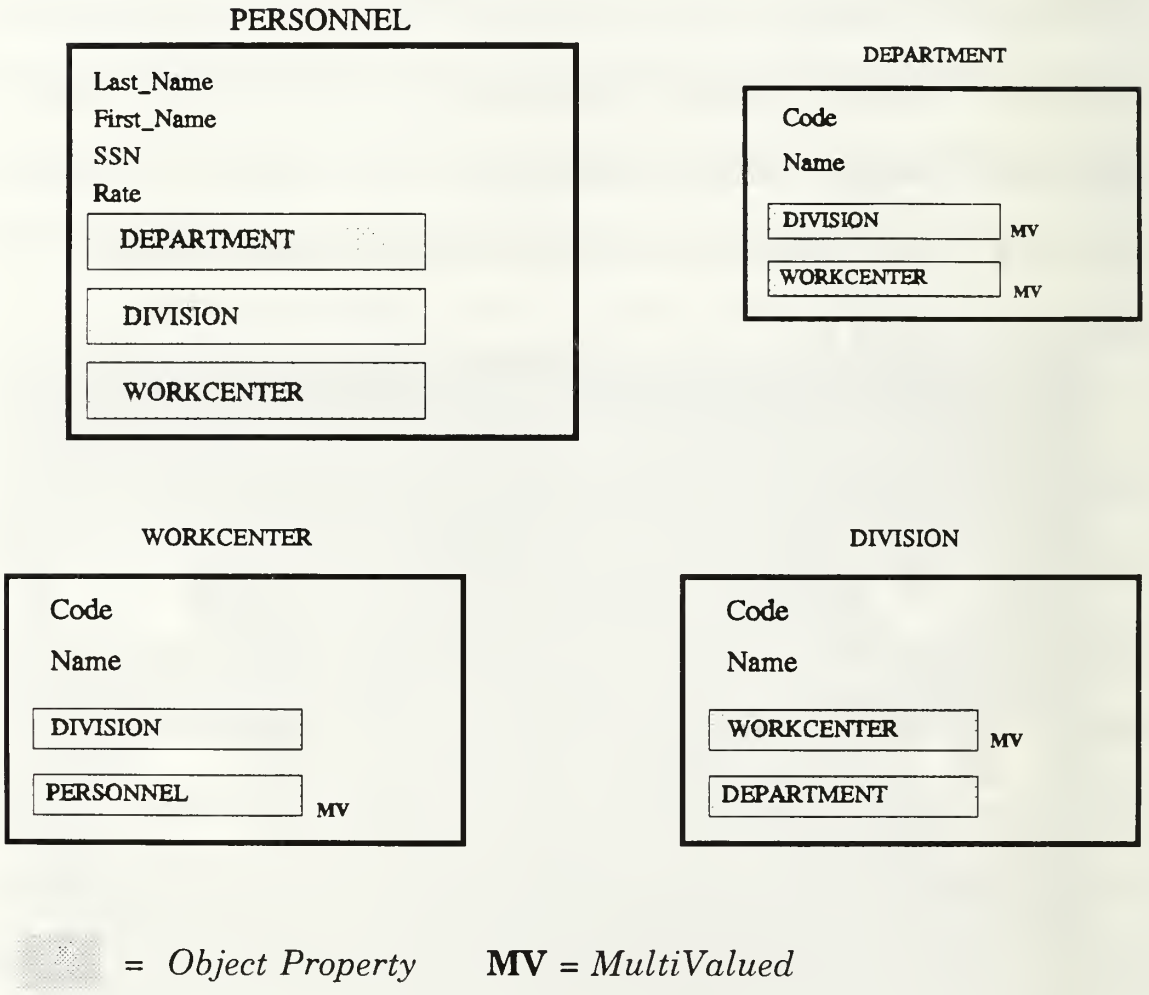
It was decided that a modern fourth-generation language would serve as the development environment. Fourth-generation languages are renown for their advantages over the more traditional programming languages, particularly in the area of rapid prototyping. The Clarion Professional Developer, version 2.1, was selected as the development platform. Clarion is designed for developing commercial-quality business application programs for microcomputers. The concepts of 4GLs in general and the Clarion Professional Developer in particular are the subject of Chapter IV.

DRUGDOG version 3.0 is the result of this study. It is a comprehensive software application that will greatly aid individual Urinalysis Coordinators in implementing the Navy's Random Urinalysis Program within their command. A hard drive installation program accompanies the package for those commands where ADP security allows the program to reside and operate from the hard drive. An exhaustive set of context-sensitive help screens and a complete, on-line user's manual will make *DRUGDOG* easy for even the computer neophyte to the use. The final product of this thesis provides an actual working tool for the fleet.

APPENDIX A

REQUIREMENTS DOCUMENTATION

A. TABLE 1: SEMANTIC OBJECT DIAGRAM



= *Object Property* **MV** = *MultiValued*

B. TABLE 2: OBJECT DEFINITIONS

PERSONNEL OBJECT

Last_Name; Personnel-last-name

First_Name; Personnel-first-name

SSN; Personnel-SSN

Rate; Personnel-Rate

WORKCENTER; WORKCENTER object

DIVISION; DIVISION object

DEPARTMENT; DEPARTMENT object

WORKCENTER OBJECT

Code; Workcenter-Code

Name; Workcenter-Name

DIVISION; DIVISION object

PERSONNEL; PERSONNEL object; MV

DEPARTMENT OBJECT

Code; Department-Code

Name; Department-Name

DIVISION; DIVISION object

WORKCENTER; WORKCENTER object; MV

DIVISION OBJECT

Code; Division-Code

Name; Division-Name

WORKCENTER; WORKCENTER object; MV

DEPARTMENT; DEPARTMENT object

C. TABLE 3: DOMAIN DEFINITIONS

Personnel-Last-Name:
Text 20
Last name of individual personnel

Personnel-First-Name:
Text 15
First name of individual personnel

Personnel-SSN:
Numeric 11, mask NNN-NN-NNNN
Social Security Number of individual personnel
Mandatory Privacy Act compliance

Personnel-Rate:
Text 6
Military rate of individual personnel

Workcenter-Code:
Text 3
Unique code for each workcenter

Workcenter-Name:
Text 12
Formal name of a Workcenter

Division-Code:
Text 3
Unique code for each Division

Division-Name:
Text 10
Formal name of a Division

Department-Code:
Text 3
Unique code for each Department

Department-Name:
Text 10
Formal name of a Division

D. TABLE 4: PERSONNEL UPDATE MECHANISMS

A. Add new PERSONNEL data
1. Inputs
* Listing of new personnel from Workcenter LPO
2. Outputs
* New PERSONNEL object instance in database
3. Processing notes
* Workcenter data may not be available upon checkin
4. Volume
* Up to 3000 personnel
5. Frequency
* Ongoing basis

B. Delete PERSONNEL data
1 Inputs
* Listing of personnel to delete from LPO
* PERSONNEL objects in database
2. Outputs
*Confirmation on screen
3. Processing notes
Backups of PERSONNEL data should be made prior to record deletion.
4. Volume
* Approximately 10 to 20 personnel depending on command size
5. Frequency
* Delete monthly

C. Modify PERSONNEL data
1. Inputs
* PERSONNEL object instance from database
* Includes properties of WORKCENTER object
2. Outputs
* Modified object instance to database

* Confirmation on screen

3. Processing notes
   * This function changes properties of PERSONNEL
4. Volume
   * Up to 3000 personnel
5. Frequency
   * Weekly/Monthly

E. TABLE 5: PERSONNEL DISPLAY AND CONTROL MECHANISMS

A. Query on PERSONNEL
   1. Output description
      * Form that shows all pertinent data on personnel
   2. Source data
      * PERSONNEL object
      * Personnel name keyed by Urinalysis Coordinator
   3. Processing notes
      * Used by Urinalysis Coordinator
   4. Volume
      * Command-driven
   5. Frequency
      * Weekly

B. PERSONNEL Reports
   1. Output description
      * Report showing PERSONNEL data
   2. Source data
      * PERSONNEL object
      * Personnel name or SSN keyed by Urinalysis Coordinator
   3. Processing notes
      * Used by Workcenter Urinalysis Coordinator
   4. Volume
      * Command-driven
   5. Frequency
      * As required

C. PERSONNEL Control Mechanisms

1. Access
   * Provide password system to ensure that only the Urinalysis Coordinator can execute the system

2. Procedures
   * Establish local procedures to ensure accuracy and integrity of PERSONNEL objects in data store

APPENDIX B

DATA DICTIONARY

A. TABLE 6: *DRUGDOG*.DAT DATA ELEMENTS

ELEMENT	TYPE	WIDTH	DESCRIPTION
Last_Name	Char	20	Personnel's last name
First_name	Char	15	Personnel's first name
SSN	Long	9	Social Security Number
Rate	Char	6	Military service rating
WorkCenter	Char	10	Name of workcenter
Div	Char	10	Name of division
Dept	Char	10	Name of department
Quota	Long	3	Number of desired selections
TotalRecs	Long	5	Total size of command roster

B. TABLE 7: *DRUGDOG* SYSTEM VIEWS

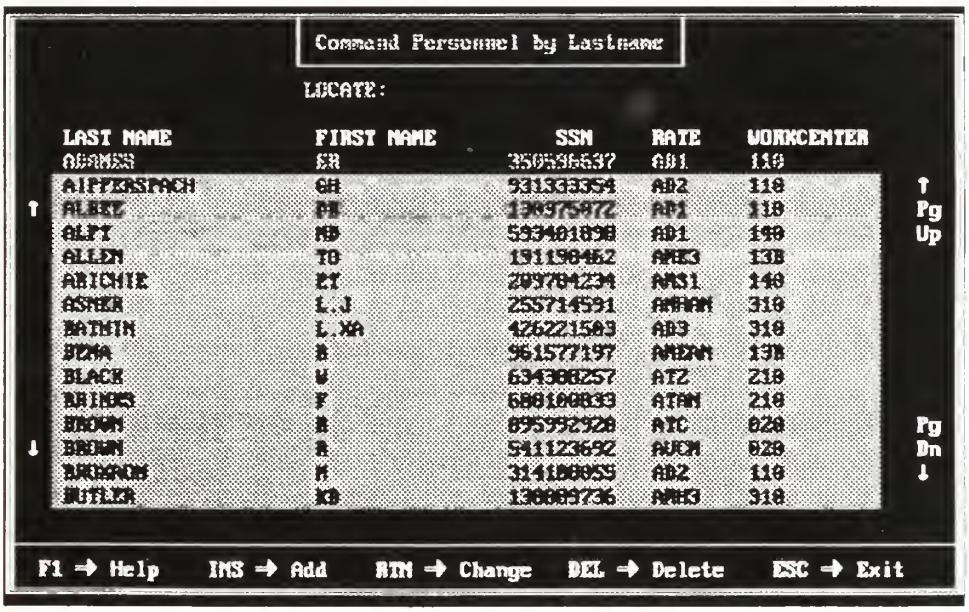
1. Display of Total Personnel

View File: DRUGDOG.DAT

Data Elements: Last_Name, First_Name, SSN, Rate, WorkCenter

Description: All personnel records can be browsed via a highlight bar in ascending alphabetical order. Records can be selected for deletion or modification by placing the bar on the desired record and pressing <RTN>. A new record can be added by pressing the insert key <INS> at any point during browsing. All tables are designed in the same manner, with the same commands available.

Example:



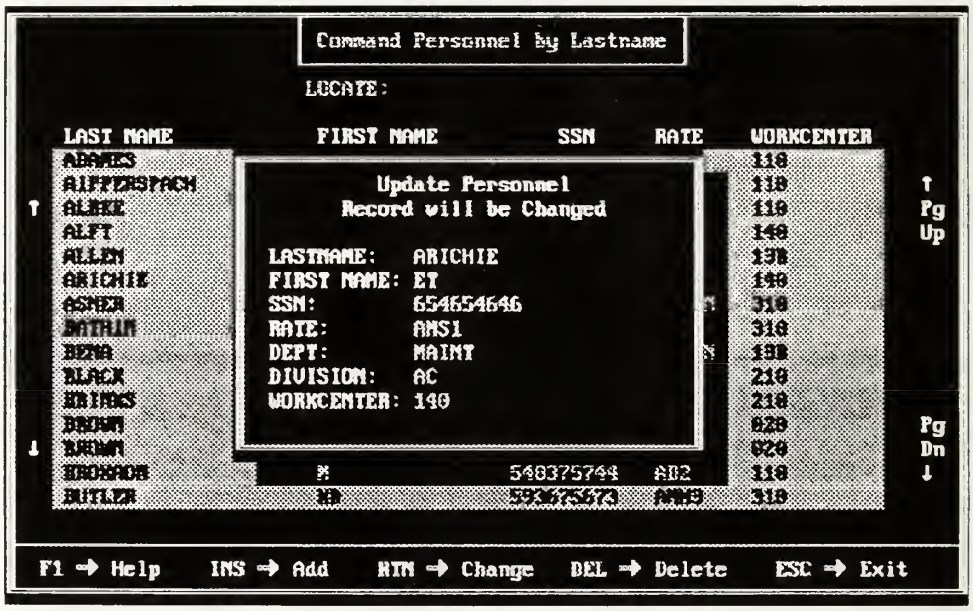
2. Record Addition/Modification Form

View Files: DRUGDOG.DAT, SHOWFORM.CLA

Data Elements: Last_Name, First_Name, SSN, Rate, WorkCenter, Division, Department

Description: All data fields can be modified using this form. It is identical for all browse tables.

Example:



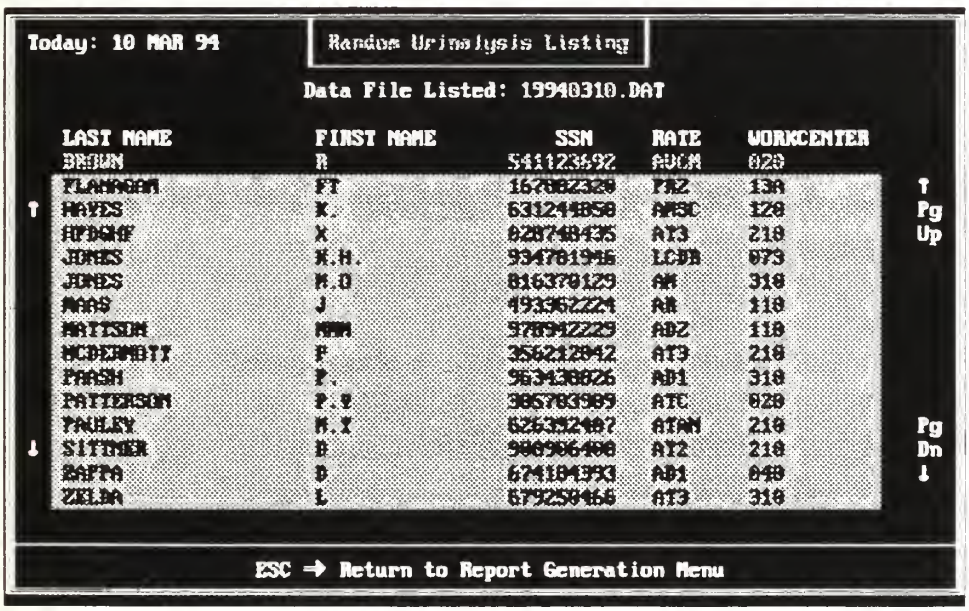
3. Display of a Random Urinalysis Selection

View Files: DRUGDOG.DAT, URINTABL.CLA

Data Elements: Last_Name, First_Name, SSN, Rate, WorkCenter

Description: This displays personnel selected for Urinalysis. This table is read-only; no modifications are allowed.

Example:



C. TABLE 8: *DRUGDOG* REPORTS

1. Urinalysis Listings

Report Files: DRUGDOG.DAT, SHOWRAND.CLA

Data Elements:Last_Name, First_Name, SSN, Rate

Description: This report prints those personnel who were selected for Urinalysis. They can be printed by Workcenter, Division or Department order. A blank line is provided for member signature.

Example:

March 1994 Urinalysis List for WorkCenter 020

Name	Rate	SSN	Signature
Archry, David	AT3	183732788	_____
Bena, Brian	AME2	184796767	_____
Farver, Mike	AMH1	867938047	_____
Hanigan, Fred	LCDR	903758288	_____
Tracks, Jerry	AN	957436332	_____
Vavra, Frenando	AD3	773732788	_____

Roster Listings Report Files:DRUGDOG.DAT, ROSTER.CLA

Data Elements:Last_Name, First_Name, SSN, Rate,
options: Workcenter, Div, or Dept

Description: This report prints alphabetic lists of all personnel in the command. They can be printed by Workcenter, Division, or Department. These are used to ensure the database is current.

Example:

March 1994 Command Roster for Aircraft Division

NAME	RATE	SSN	WORKCENTER
Archry, David	AN	183732788	Powerplants
Bena, Brian	AME2	184796767	Seat Shop
Farver, Mike	AMH1	867938047	Airframes
Hanigan, Fred	LCDR	903758288	A/C office
Tracks, Jerry	AN	957436332	Airframes
Vavra, Frenando	AD3	773732788	Powerplants

APPENDIX C

DRUGDOG 3.0 SOURCE CODE

```
DRUGDOG          PROGRAM
         INCLUDE('STD_KEYS.CLA')
         INCLUDE('CTL_KEYS.CLA')
         INCLUDE('ALT_KEYS.CLA')
         INCLUDE('SHF_KEYS.CLA')

REJECT_KEY   EQUATE(CTRL_ESC)
ACCEPT_KEY   EQUATE(CTRL_ENTER)
TRUE         EQUATE(1)
FALSE        EQUATE(0)

         MAP
          PROC(G_OPENFILES)
          PROC(G_OPENFILES2)
          MODULE('DRUGDO01')
            PROC(MAINMENU) !USN Random Urinalysis Program
          .
          MODULE('DRUGDO02')
            PROC(SHO_LASTNAME) !Show Drugdog By Lastname
          .
          MODULE('DRUGDO03')
            PROC(UPD_DRUGDOG) !Update Drugdog
          .
          MODULE('DRUGDO04')
            PROC(PERS_MENU) !Personnel Roster Maintenance
          .
          MODULE('DRUGDO05')
            PROC(SHO_DEPT) !Show Drugdog By Department
          .
          MODULE('DRUGDO06')
            PROC(SHO_DIV) !Show Drugdog By Division
          .
          MODULE('DRUGDO07')
            PROC(SHO_WORKCENT) !Show Drugdog By Workcenter
```

```
.
MODULE('DRUGDO08')
  PROC(OBTAIN_DEPT) !Obtain desired Dept for filter
.
MODULE('DRUGDO09')
  PROC(QUOTA)    !Quota for Random Urinalysis
.
MODULE('DRUGDO10')
  PROC(PRINT_RPT) !Report Generation Menu
.
MODULE('DRUGDO11')
  PROC(GET_DATAFILE) !Obtain data file to display
.
MODULE('DRUGDO12')
  PROC(SHOW_RANDS) !Show Urinalysis Listing
.
MODULE('DRUGDO13')
  PROC(SHOW_NEWLIST) !Show Urinalysis Listing
.
MODULE('DRUGDO14')
  PROC(OBTAIN_DIV) !Obtain desired Div for filter
.
MODULE('DRUGDO15')
  PROC(OBTAIN_WC) !Obtain desired W/C for filter
.
MODULE('DRUGDO16')
  PROC(PRT_RANDS)
.
MODULE('DRUGDO17')
  PROC(PRT_ROSTER)
.
MODULE('READ_MAN')
  PROC(MANUAL) !Displays User's Guide
.
MODULE('GET_RAND')
  PROC(GET_RAND) !RANDOM SELECTION PROCESS
.
MODULE('PRESS')
  PROC(PRESS)    !KEYSTROKE DELAY
.

.
EJECT('FILE LAYOUTS')
DRUGDOG          FILE,PRE(DRU),CREATE,RECLAIM
BY_LASTNAME   KEY(DRU:LASTNAME),DUP,NOCASE,OPT
BY_DEPT       KEY(DRU:DEPT,DRU:LASTNAME),DUP,NOCASE,OPT
BY_DIV        KEY(DRU:DIV,DRU:LASTNAME),DUP,NOCASE,OPT
BY_WORKCENT   KEY(DRU:WORKCENT,DRU:LASTNAME),DUP,NOCASE,OPT
RECORD        RECORD
FIRST_NAME     STRING(15)
```

```
SSN          LONG
DEPT              STRING(10)
DIV          STRING(10)
WORKCENT     STRING(10)
RATE              STRING(6)
LASTNAME     STRING(20)
          . .

TEMPRAND     FILE,PRE(TEM),CREATE,RECLAIM
BY_LASTNAME    KEY(TEM:LASTNAME),DUP,NOCASE,OPT
BY_DEPT      KEY(TEM:DEPT,TEM:LASTNAME),DUP,NOCASE,OPT
BY_DIV       KEY(TEM:DIV,TEM:LASTNAME),DUP,NOCASE,OPT
BY_WORKCENT    KEY(TEM:WORKCENT,TEM:LASTNAME),DUP,NOCASE,OPT
RECORD       RECORD
FIRST_NAME     STRING(15)
SSN          LONG
DEPT              STRING(10)
DIV          STRING(10)
WORKCENT     STRING(10)
RATE              STRING(6)
LASTNAME     STRING(20)
          . .

          EJECT('GLOBAL MEMORY VARIABLES')
ACTION         SHORT          !0 = NO ACTION
                              !1 = ADD RECORD
                              !2 = CHANGE RECORD
                              !3 = DELETE RECORD
                              !4 = LOOKUP FIELD
          GROUP,PRE(MEM)
MESSAGE        STRING(30)  !Global Message Area
PAGE           SHORT       !Report Page Number
LINE           SHORT       !Report Line Number
DEVICE         STRING(30)  !Report Device Name
GET_DEPT       STRING(10)  !Obtain Department filter
GET_DIV        STRING(10)  !Obtain desired division
GET_WORKCENT   STRING(10) !Obtain Workcenter for filter
DATA_FILE      STRING(12)  !Data file of random selections
QUOTA_NUMBER   LONG        !NUMBER OF DESIRED SELECTIONS
TOTAL_RECS     LONG        !Total # of Personnel
REPORT_TYPE    STRING(1) !What type of report desired?
          .

          EJECT('CODE SECTION')
 CODE
 SETHUE(7,0)                !SET WHITE ON BLACK
 BLANK                      !  AND BLANK
 HELP('DRUGDOG.HLP')        !OPEN THE HELP FILE
 G_OPENFILES                !OPEN OR CREATE FILES
```

```
  SETHUE()                    !   THE SCREEN
  MAINMENU                    !USN Random Urinalysis Program
  RETURN                      !EXIT TO DOS

G_OPENFILES  PROCEDURE !OPEN FILES & CHECK FOR ERROR
  CODE
  G_OPENFILES2(DRUGDOG) !CALL OPEN FILE PROCEDURE
  G_OPENFILES2(TEMPRAND) !CALL OPEN FILE PROCEDURE
  BLANK                       !BLANK THE SCREEN

G_OPENFILES2 PROCEDURE(G_FILE) !OPEN EACH FILE & CHECK ERROR
G_FILE       EXTERNAL,FILE
FILE_NAME   STRING(64)

  CODE
  FILE_NAME = NAME(G_FILE)
  SHOW(25,1,CENTER('OPENING FILE: ' & CLIP(FILE_NAME),80)) !DISPLAY FILE
NAME
  OPEN(G_FILE)                !OPEN THE FILE
  IF ERROR()                  !OPEN RETURNED AN ERROR
    CASE ERRORCODE()          ! CHECK FOR SPECIFIC ERROR
    OF 46                     !  KEYS NEED TO BE REBUILT
      SETHUE(0,7)             !  BLACK ON WHITE
      SHOW(25,1,CENTER('REBUILDING KEY FILES FOR ' & CLIP(FILE_NAME),80))
      BUILD(G_FILE)           !  CALL THE BUILD PROCEDURE
      IF ERROR()              !  ON UNNEXPECTED ERROR
        LOOP                  !  STOP EXECUTION
          STOP('Cannot Build ' & FILE_NAME & ' - Error: ' & ERROR())
        .
      .
      SETHUE(7,0)             !  WHITE ON BLACK
      BLANK(25,1,1,80)        !  BLANK THE MESSAGE
    OF 2                      !IF NOT FOUND,
      CREATE(G_FILE)          ! CREATE
      IF ERROR()              !STOP ON UNNEXPECTED ERROR
        LOOP
          STOP('Cannot Create ' & FILE_NAME & ' - Error: ' & ERROR())
      . .
    OF 73                     ! MEMO FILE NOT FOUND
      LOOP                    !  STOP EXECUTION
        STOP('Cannot Open Memo File for ' & FILE_NAME & ERROR())
      .
    ELSE                      ! ANY OTHER ERROR
      LOOP                    !  STOP EXECUTION
        STOP('Cannot Open ' & FILE_NAME & ' - Error: ' & ERROR())
      .
  . .

*************************************************************************
```

```
        MEMBER('DRUGDOG')
MAINMENU    PROCEDURE

SCREEN      SCREEN        PRE(SCR),WINDOW(25,80),AT(1,1),HUE(15,1)
          ROW(10,26) PAINT(12,30),HUE(15,4)
          ROW(22,27) PAINT(1,30),HUE(15,0)
          ROW(8,57)  PAINT(15,2),HUE(15,0)
          ROW(6,25)  PAINT(4,32),HUE(15,4)
          ROW(23,27) PAINT(1,34),HUE(7,1)
          ROW(1,1)     STRING('╔═{31}╦═{15}╦═{30}╗'),HUE(15,1)
          ROW(2,1)     STRING('‖<0{31}>‖<0{15}>‖<0{30}>‖'),HUE(15,1)
          ROW(3,1)     STRING('‖<0{31}>╚═{15}╝<0{30}>‖'),HUE(15,1)
          ROW(4,1)     REPEAT(4);STRING('‖<0{78}>‖'),HUE(15,1) .
          ROW(8,1)     STRING('‖<0{5}>███ <0,0>██<0{49}>██<0{6}>██<0{7}>‖')  |
                 HUE(15,1)
          ROW(9,1)     STRING('‖<0{6}>███ <0>██<0{50}>██<0{4}>██<0{8}>‖')    |
                 HUE(15,1)
          ROW(10,1)  STRING('‖<0{7}>██<0>██<0{51}>██<0,0>██<0{9}>‖')        |
                 HUE(15,1)
          ROW(11,1)  STRING('‖<0{7}> ██████<0{52}>█████<0{10}>‖'),HUE(15,1)
          ROW(12,1)  REPEAT(3);STRING('‖<0{78}>‖'),HUE(15,1) .
          ROW(15,1)  STRING('‖<0{10}> ████ <0{51}>██<0{4}>██<0{4}>‖')       |
                 HUE(15,1)
          ROW(16,1)  STRING('‖<0{8}> ▐{7}▌ <0{49}>██<0{4}>██<0{4}>‖')       |
                 HUE(15,1)
          ROW(17,1)  STRING('‖<0{7}>████<0{5}>████<0{48}>▐{8}<0{4}>‖')      |
                 HUE(15,1)
          ROW(18,1)  STRING('‖<0{7}>▐{11}<0{51}>██<0{7}>‖'),HUE(15,1)
          ROW(19,1)  STRING('‖<0{7}>██<0{7}>██<0{51}>██<0{7}>‖'),HUE(15,1)
          ROW(20,1)  REPEAT(5);STRING('‖<0{78}>‖'),HUE(15,1) .
          ROW(25,1)  STRING('╚═{78}╝'),HUE(15,1)
          ROW(6,25)  STRING('╔═{5}╤═{18}╤═{5}╗'),HUE(15,4)
          ROW(7,25)  STRING('‖<0{5}>│<0{18}>│<0{5}>‖'),HUE(15,4)
          ROW(8,5)     STRING('▐██<0{18}>‖<0{5}>└─{18}┘<0{5}>‖'),HUE(15,4)
          ROW(9,5)     REPEAT(3);STRING('▐██<0{18}>‖<0{30}>‖'),HUE(15,4) .
          ROW(12,25) REPEAT(9);STRING('‖<0{30}>‖'),HUE(15,4) .
          ROW(21,25) STRING('╨═{30}╜'),HUE(15,4)
          ROW(24,11) STRING('=<16,0{20},24,0,25>'),HUE(14,1)
          ROW(2,34)  STRING(' DRUGDOG v 3.0 '),HUE(11,0)
          ROW(4,17)  STRING('U')
            COL(18)  STRING('NITED '),HUE(15,1)
            COL(25)  STRING('S')
            COL(26)  STRING('TATES '),HUE(15,1)
            COL(33)  STRING('N')
            COL(34)  STRING('AVY '),HUE(15,1)
            COL(39)  STRING('R')
            COL(40)  STRING('ANDOM'),HUE(15,1)
            COL(46)  STRING(' '),HUE(15,1)
            COL(47)  STRING('U')
```

```
          COL(48)  STRING('RINALYSIS  PROGRAM'),HUE(15,1)
         ROW(7,33)  STRING('M A I N  M E N U'),HUE(14,4)
         ROW(24,8)  STRING('F1'),HUE(14,1)
          COL(14)  STRING('Help'),HUE(14,1)
          COL(34)  STRING(' '),HUE(14,1)
          COL(37)  STRING(' Select Option'),HUE(14,1)
          COL(62)  STRING('Today:'),HUE(14,1)DATE
          COL(69)  STRING(@D7),HUE(14,1)
                ENTRY,USE(?FIRST_FIELD)
                ENTRY,USE(?PRE_MENU)
                MENU,USE(MENU_FIELD"),REQ
         ROW(10,29)   STRING('Conduct Random Urinalysis'),HUE(15,4)      |
                 SEL(14,0)
         ROW(12,33)   STRING('Roster Maintenance'),HUE(15,4),SEL(14,0)
         ROW(14,35)   STRING('Print Reports'),HUE(15,4),SEL(14,0)
         ROW(16,32)   STRING('Database Utilities'),HUE(15,4),SEL(14,0)
         ROW(18,35)   STRING('User"s Manual'),HUE(15,4),SEL(14,0)
         ROW(20,35)   STRING('Exit Program'),HUE(15,4),SEL(14,0)
       .          .

EJECT
CODE
OPEN(SCREEN)              !OPEN THE MENU SCREEN
SETCURSOR                 !TURN OFF ANY CURSOR
MENU_FIELD" = "           !START MENU WITH FIRST ITEM
LOOP                      !LOOP UNTIL USER EXITS
 SCR:DATE = TODAY()
 ALERT                    !TURN OFF ALL ALERTED KEYS
 ALERT(REJECT_KEY)        !ALERT SCREEN REJECT KEY
 ALERT(ACCEPT_KEY)        !ALERT SCREEN ACCEPT KEY
 ACCEPT                   !READ A FIELD OR MENU CHOICE
 IF CHUCKED() = REJECT_KEY THEN RETURN. !RETURN ON SCREEN REJECT

 IF CHUCKED() = ACCEPT_KEY !ON SCREEN ACCEPT KEY
  UPDATE                  ! MOVE ALL FIELDS FROM SCREEN
  SELECT(?)               ! START WITH CURRENT FIELD
  SELECT                  ! EDIT ALL FIELDS
  CYCLE                   ! GO TO TOP OF LOOP
 .                        !

 CASE FIELD()             !JUMP TO FIELD EDIT ROUTINE
 OF ?FIRST_FIELD          !FROM THE FIRST FIELD
  IF CHUCKED() = ESC_KEY THEN RETURN. !  RETURN ON ESC KEY

 OF ?PRE_MENU             !PRE MENU FIELD CONDITION
  IF CHUCKED() = ESC_KEY !  BACKING UP?
    SELECT(?-1)           !   SELECT PREVIOUS FIELD
  ELSE                    ! GOING FORWARD
    SELECT(?+1)           !   SELECT MENU FIELD
```

```
  OF ?MENU_FIELD"          !FROM THE MENU FIELD
    EXECUTE CHOICE()       ! CALL THE SELECTED PROCEDURE
      GET_RAND             ! RANDOM SELECTION PROCESS
      PERS_MENU            ! Personnel Roster Maintenance
      PRINT_RPT            ! Report Generation Menu
      PRT_ROSTER           !
      MANUAL               ! Displays User's Guide
      RETURN
  . . .

*****************************************************************************

          MEMBER('DRUGDOG')
SHO_LASTNAME PROCEDURE

SCREEN       SCREEN        PRE(SCR),WINDOW(25,80),AT(1,1),HUE(15,1)
             ROW(22,7)  PAINT(1,67),HUE(0,7)
             ROW(23,7)  PAINT(1,69),HUE(7,0)
             ROW(7,74)  PAINT(16,2),HUE(7,0)
             ROW(22,5)  PAINT(1,1),HUE(7,1)
             ROW(22,6)  PAINT(1,1),HUE(0,1)
             ROW(7,5)     PAINT(15,69),HUE(0,7)
             ROW(1,1)     STRING('╔═{23}╤═{31}╤═{22}╗'),HUE(15,1)
             ROW(2,1)     STRING('‖<0{23}>|<0{31}>|<0{22}>‖'),HUE(15,1)
             ROW(3,1)     STRING('‖<0{23}>└─{31}┘<0{22}>‖'),HUE(15,1)
             ROW(4,1)     REPEAT(4);STRING('‖<0{78}>‖'),HUE(15,1) .
             ROW(8,1)     STRING('‖<0{75},24,0,0>‖'),HUE(15,1)
             ROW(9,1)     STRING('‖<0,24,0{76}>‖'),HUE(15,1)
             ROW(10,1)  REPEAT(9);STRING('‖<0{78}>‖'),HUE(15,1) .
             ROW(19,1)  STRING('‖<0,25,0{76}>‖'),HUE(15,1)
             ROW(20,1)  STRING('‖<0{75},25,0,0>‖'),HUE(15,1)
             ROW(21,1)  REPEAT(2);STRING('‖<0{78}>‖'),HUE(15,1) .
             ROW(23,1)  STRING('╟─{78}╢'),HUE(15,1)
             ROW(24,1)  STRING('‖<0{78}>‖'),HUE(15,1)
             ROW(25,1)  STRING('└─{78}┘'),HUE(15,1)
             ROW(22,7)  STRING('-{69}'),HUE(0,0)
             ROW(24,7)  STRING('=<16,0{13}>=<16,0{12}>=<16,0{14}>=<16>'      |
                      & '<0{15}>=<16>'),HUE(14,1)
             ROW(2,26)  STRING(' Command Personnel by Lastname '),HUE(11,0)
             ROW(4,26)  STRING('LOCATE:'),HUE(11,1)
             ROW(6,6)     STRING('LAST NAME {12}FIRST NAME {10}SSN {5}RATE')
               COL(63)  STRING('WORKCENTER'),HUE(15,1)
             ROW(9,77)  STRING('Pg')
             ROW(10,77) STRING('Up')
             ROW(18,77) STRING('Pg')
             ROW(19,77) STRING('Dn')
             ROW(24,4)  STRING('F1'),HUE(14,4)
               COL(9)     STRING(' Help'),HUE(14,1)
```

```
            COL(17)  STRING(' '),HUE(14,1)
            COL(18)  STRING('INS'),HUE(14,4)
            COL(25)  STRING('Add'),HUE(14,1)
            COL(32)  STRING('RTN'),HUE(14,4)
            COL(39)  STRING('Change  '),HUE(14,1)
            COL(48)  STRING('DEL'),HUE(14,4)
            COL(55)  STRING('Delete  '),HUE(14,1)
            COL(65)  STRING('ESC'),HUE(14,4)
            COL(72)  STRING('Exit'),HUE(14,1)
LOCATOR       ROW(4,34)  STRING(20),HUE(14,0)
                ENTRY,USE(?FIRST_FIELD)
                ENTRY,USE(?PRE_POINT)
                REPEAT(15),EVERY(1),INDEX(NDX)
          ROW(7,5)       POINT(1,69),USE(?POINT),ESC(?-1)
LASTNAME      COL(6)        STRING(20)
FIRST_NAME    COL(27)   STRING(15)
SSN         COL(43)   STRING(@N09)
RATE          COL(55)   STRING(6)
WORKCENT      COL(63)   STRING(10)
          .         .

NDX     BYTE                !REPEAT INDEX FOR POINT AREA
ROW           BYTE          !ACTUAL ROW OF SCROLL AREA
COL     BYTE                !ACTUAL COLUMN OF SCROLL AREA
COUNT         BYTE(15)      !NUMBER OF ITEMS TO SCROLL
ROWS          BYTE(15)      !NUMBER OF ROWS TO SCROLL
COLS          BYTE(69)      !NUMBER OF COLUMNS TO SCROLL
FOUND         BYTE          !RECORD FOUND FLAG
NEWPTR        LONG          !POINTER TO NEW RECORD

TABLE         TABLE,PRE(TBL) !TABLE OF RECORD DATA
PTR       LONG              !  POINTER TO FILE RECORD
LASTNAME      STRING(20)
FIRST_NAME    STRING(15)
SSN       LONG
RATE            STRING(6)
WORKCENT      STRING(10)
          .

  EJECT
  CODE
  ACTION# = ACTION          !SAVE ACTION
  OPEN(SCREEN)              !OPEN THE SCREEN
  SETCURSOR                 !TURN OFF ANY CURSOR
  TBL:PTR = 1               !START AT TABLE ENTRY
  NDX = 1                   !PUT SELECTOR BAR ON TOP ITEM
  ROW = ROW(?POINT)         !REMEMBER TOP ROW AND
  COL = COL(?POINT)         !LEFT COLUMN OF SCROLL AREA
  RECORDS# = TRUE           !INITIALIZE RECORDS FLAG
```

```
 CACHE(DRU:BY_LASTNAME,.25) !CACHE KEY FILE
 IF ACTION = 4              ! TABLE LOOKUP REQUEST
  NEWPTR = POINTER(DRUGDOG) ! SET POINTER TO RECORD
  IF NOT NEWPTR             ! RECORD NOT PASSED TO TABLE
   SET(DRU:BY_LASTNAME,DRU:BY_LASTNAME) !     POSITION TO CLOSEST
RECORD
   NEXT(DRUGDOG)            !   READ RECORD
   NEWPTR = POINTER(DRUGDOG) !   SET POINTER

  .
  DO FIND_RECORD            ! POSITION FILE
 ELSE
  NDX = 1                   ! PUT SELECTOR BAR ON TOP ITEM
  DO FIRST_PAGE             ! BUILD MEMORY TABLE OF KEYS

 .
 RECORDS# = TRUE            ! ASSUME THERE ARE RECORDS
 LOOP                       !LOOP UNTIL USER EXITS
  ACTION = ACTION#          !RESTORE ACTION
  ALERT                     !RESET ALERTED KEYS
  ALERT(REJECT_KEY)         !ALERT SCREEN REJECT KEY
  ALERT(ACCEPT_KEY)         !ALERT SCREEN ACCEPT KEY
  ACCEPT                    !READ A FIELD
  IF CHUCKED() = REJECT_KEY THEN BREAK. !RETURN ON SCREEN REJECT
KEY
  IF CHUCKED() = ACCEPT_KEY  ! !ON SCREEN ACCEPT KEY
  AND FIELD() <> ?POINT  !BUT NOT ON THE POINT FIELD
   UPDATE                   ! MOVE ALL FIELDS FROM SCREEN
   SELECT(?)                ! START WITH CURRENT FIELD
   SELECT                   ! EDIT ALL FIELDS
   CYCLE                    ! GO TO TOP OF LOOP
  .

  CASE FIELD()              !JUMP TO FIELD EDIT ROUTINE

  OF ?FIRST_FIELD           !FROM THE FIRST FIELD
   IF CHUCKED() = ESC_KEY   ! !  RETURN ON ESC KEY
   OR RECORDS# = FALSE !  OR NO RECORDS
    BREAK                   !   EXIT PROCEDURE
   .
  OF ?PRE_POINT             !PRE POINT FIELD CONDITION
   IF CHUCKED() = ESC_KEY ! BACKING UP?
    SELECT(?-1)             !   SELECT PREVIOUS FIELD
   ELSE                     ! GOING FORWARD
    SELECT(?POINT)          !   SELECT MENU FIELD
   .
   IF CHUCKED() = ESC_KEY ! BACKING UP?
    SCR:LOCATOR = ''        !   CLEAR LOCATOR
    SETCURSOR               !   AND TURN CURSOR OFF
   ELSE                     ! GOING FORWARD
    LEN# = 0                !   RESET TO START OF LOCATOR
```

```
    SETCURSOR(ROW(SCR:LOCATOR),COL(SCR:LOCATOR)) !AND TURN CURSOR
ON
    .
  OF ?POINT                  !PROCESS THE POINT FIELD
   IF RECORDS(TABLE) = 0 !IF THERE ARE NO RECORDS
    CLEAR(DRU:RECORD) !  CLEAR RECORD AREA
    ACTION = 1               !  SET ACTION TO ADD
    GET(DRUGDOG,0)           !  CLEAR PENDING RECORD
    UPD_DRUGDOG              !  CALL FORM FOR NEW RECORD
    NEWPTR = POINTER(DRUGDOG) !   SET POINTER TO NEW RECORD
    DO FIRST_PAGE            !  DISPLAY THE FIRST PAGE
    IF RECORDS(TABLE) = 0 !  IF THERE AREN'T ANY RECORDS
     RECORDS# = FALSE  !   INDICATE NO RECORDS
     SELECT(?PRE_POINT-1) !   SELECT THE PRIOR FIELD
    .
    CYCLE                    !   AND LOOP AGAIN
   .
   IF CHUCKED() > 31           | !THE DISPLAYABLE CHARACTERS
   AND CHUCKED() < 255  !ARE USED TO LOCATE RECORDS
    IF LEN# < SIZE(SCR:LOCATOR) !  IF THERE IS ROOM LEFT
     SCR:LOCATOR = SUB(SCR:LOCATOR,1,LEN#) & CHR(CHUCKED())
     LEN# += 1               !   INCREMENT THE LENGTH
    .
   ELSIF CHUCKED() = BS_KEY !BACKSPACE UNTYPES A CHARACTER
    IF LEN# > 0              !  IF THERE ARE CHARACTERS LEFT
     LEN# -= 1               !   DECREMENT THE LENGTH
     SCR:LOCATOR = SUB(SCR:LOCATOR,1,LEN#) !          ERASE   THE   LAST
CHARACTER
    .
   ELSE                       !FOR ANY OTHER CHARACTER
    LEN# = 0                 !  ZERO THE LENGTH
    SCR:LOCATOR = ''         !  ERASE THE LOCATOR FIELD
   .
   SETCURSOR(ROW(SCR:LOCATOR),COL(SCR:LOCATOR)+LEN#) !AND RESET THE
CURSOR
   DRU:LASTNAME = CLIP(SCR:LOCATOR) !   UPDATE THE KEY FIELD
   TIME# = CLOCK()            !SAVE THE TIME
   LOOP UNTIL KEYBOARD() !WAIT FOR A KEYSTROKE
    IF CLOCK() > TIME# + 50 THEN BREAK. !  BUT ONLY 1/2 OF A SECOND
   .
   IF KEYBOARD() > 31          | !THE DISPLAYABLE CHARACTERS
   AND KEYBOARD() < 255        | !ARE USED TO LOCATE RECORDS
   OR KEYBOARD() = BS_KEY !INCLUDE BACKSPACE
    CYCLE
   .
   IF LEN# > 0                !ON A LOCATOR REQUEST
    DRU:LASTNAME = CLIP(SCR:LOCATOR) !   UPDATE THE KEY FIELD
    SET(DRU:BY_LASTNAME,DRU:BY_LASTNAME) !  POINT TO NEW RECORD
    NEXT(DRUGDOG)            !  READ A RECORD
```

```
    IF (EOF(DRUGDOG) AND ERROR()) !  IF EOF IS REACHED
      SET(DRU:BY_LASTNAME) !    SET TO FIRST RECORD
      PREVIOUS(DRUGDOG) !    READ THE LAST RECORD
    .
    NEWPTR = POINTER(DRUGDOG) !  SET NEW RECORD POINTER
    SKIP(DRUGDOG,-1)      ! BACK UP TO FIRST RECORD
    FREE(TABLE)           ! CLEAR THE TABLE
    DO NEXT_PAGE          ! AND DISPLAY A NEW PAGE
  .
  CASE CHUCKED()          !PROCESS THE KEYSTROKE

  OF INS_KEY              !INS KEY
    CLEAR(DRU:RECORD) !  CLEAR RECORD AREA
    ACTION = 1            ! SET ACTION TO ADD
    GET(DRUGDOG,0)        ! CLEAR PENDING RECORD
    UPD_DRUGDOG           ! CALL FORM FOR NEW RECORD
    IF ~ACTION            ! IF RECORD WAS ADDED
      NEWPTR = POINTER(DRUGDOG) !    SET POINTER TO NEW RECORD
      DO FIND_RECORD      !    POSITION IN FILE
    .
  OF ENTER_KEY            !ENTER KEY
  OROF ACCEPT_KEY         !CTRL-ENTER KEY
    DO GET_RECORD         ! GET THE SELECTED RECORD
    IF ACTION = 4 AND CHUCKED() = ENTER_KEY!     IF THIS IS A LOOKUP
REQUEST
      ACTION = 0          !    SET ACTION TO COMPLETE
      BREAK               !    AND RETURN TO CALLER
    .
    IF ~ERROR()           ! IF RECORD IS STILL THERE
      ACTION = 2          !    SET ACTION TO CHANGE
      UPD_DRUGDOG         !    CALL FORM TO CHANGE REC
      IF ACTION           !    IF SUCCESSFUL RE-DISPLAY
        ACTN# = 0 THEN CYCLE.
    .
    NEWPTR = POINTER(DRUGDOG) !    SET POINTER TO NEW RECORD
    DO FIND_RECORD        !    POSITION IN FILE
  OF DEL_KEY              !DEL KEY
    DO GET_RECORD         ! READ THE SELECTED RECORD
    IF ~ERROR()           ! IF RECORD IS STILL THERE
      ACTION = 3          !    SET ACTION TO DELETE
      UPD_DRUGDOG         !    CALL FORM TO DELETE
      IF ~ACTION          !    IF SUCCESSFUL
        N# = NDX          !SAVE POINT INDEX
        DO SAME_PAGE      !RE-DISPLAY
        NDX = N#          !RESTORE POINT INDEX
    . .
  OF DOWN_KEY             !DOWN ARROW KEY
    DO SET_NEXT           ! POINT TO NEXT RECORD
    DO FILL_NEXT          ! FILL A TABLE ENTRY
```

```
      IF FOUND                ! FOUND A NEW RECORD
        SCROLL(ROW,COL,ROWS,COLS,ROWS(?POINT))! SCROLL THE SCREEN UP
        GET(TABLE,RECORDS(TABLE)) ! GET RECORD FROM TABLE
        DO FILL_SCREEN        ! DISPLAY ON SCREEN
      .

    OF PGDN_KEY               !PAGE DOWN KEY
      DO SET_NEXT             ! POINT TO NEXT RECORD
      DO NEXT_PAGE            ! DISPLAY THE NEXT PAGE

    OF CTRL_PGDN              !CTRL-PAGE DOWN KEY
      DO LAST_PAGE            ! DISPLAY THE LAST PAGE
      NDX = RECORDS(TABLE) ! POSITION POINT BAR

    OF UP_KEY                 !UP ARROW KEY
      DO SET_PREV             ! POINT TO PREVIOUS RECORD
      DO FILL_PREV            ! FILL A TABLE ENTRY
      IF FOUND                ! FOUND A NEW RECORD
        SCROLL(ROW,COL,ROWS,COLS,-(ROWS(?POINT)))! SCROLL THE SCREEN
DOWN
        GET(TABLE,1)          ! GET RECORD FROM TABLE
        DO FILL_SCREEN        ! DISPLAY ON SCREEN
      .

    OF PGUP_KEY               !PAGE UP KEY
      DO SET_PREV             ! POINT TO PREVIOUS RECORD
      DO PREV_PAGE            ! DISPLAY THE PREVIOUS PAGE

    OF CTRL_PGUP              !CTRL-PAGE UP
      DO FIRST_PAGE           ! DISPLAY THE FIRST PAGE
      NDX = 1                 ! POSITION POINT BAR
  . . .
  FREE(TABLE)                 !FREE MEMORY TABLE
  FREE(DRU:BY_LASTNAME) !FREE CACHE
  RETURN                      !AND RETURN TO CALLER

SAME_PAGE ROUTINE             !DISPLAY THE SAME PAGE
  GET(TABLE,1)                ! GET THE FIRST TABLE ENTRY
  DO FILL_RECORD              ! FILL IN THE RECORD
  SET(DRU:BY_LASTNAME,DRU:BY_LASTNAME,TBL:PTR) ! POSITION FILE
  FREE(TABLE)                 ! EMPTY THE TABLE
  DO NEXT_PAGE                ! DISPLAY A FULL PAGE

FIRST_PAGE ROUTINE            !DISPLAY FIRST PAGE
  FREE(TABLE)                 ! EMPTY THE TABLE
  CLEAR(DRU:RECORD,-1)        ! CLEAR RECORD TO LOW VALUES
  CLEAR(TBL:PTR)              ! ZERO RECORD POINTER
  SET(DRU:BY_LASTNAME)        ! POINT TO FIRST RECORD
  LOOP NDX = 1 TO COUNT ! FILL UP THE TABLE
```

```
    DO FILL_NEXT              !   FILL A TABLE ENTRY
    IF NOT FOUND THEN BREAK. !   GET OUT IF NO RECORD
  .
  NDX = 1                     ! SET TO TOP OF TABLE
  DO SHOW_PAGE                ! DISPLAY THE PAGE

LAST_PAGE ROUTINE             !DISPLAY LAST PAGE
  NDX# = NDX                  ! SAVE SELECTOR POSITION
  FREE(TABLE)                 ! EMPTY THE TABLE
  CLEAR(DRU:RECORD,1)         ! CLEAR RECORD TO HIGH VALUES
  CLEAR(TBL:PTR,1)            ! CLEAR PTR TO HIGH VALUE
  SET(DRU:BY_LASTNAME)        ! POINT TO FIRST RECORD
  LOOP NDX = COUNT TO 1 BY -1 ! FILL UP THE TABLE
    DO FILL_PREV              !   FILL A TABLE ENTRY
    IF NOT FOUND THEN BREAK. !   GET OUT IF NO RECORD
  .                           ! END OF LOOP
  NDX = NDX#                  ! RESTORE SELECTOR POSITION
  DO SHOW_PAGE                ! DISPLAY THE PAGE

FIND_RECORD ROUTINE           !POSITION TO SPECIFIC RECORD
  SET(DRU:BY_LASTNAME,DRU:BY_LASTNAME,NEWPTR) !POSITION FILE
  IF NEWPTR = 0               !NEWPTR NOT SET
    NEXT(DRUGDOG)             ! READ NEXT RECORD
    NEWPTR = POINTER(DRUGDOG) ! SET NEWPTR
    SKIP(DRUGDOG,-1)          ! BACK UP TO DISPLAY RECORD
  .
  FREE(TABLE)                 ! CLEAR THE RECORD
  DO NEXT_PAGE                ! DISPLAY A PAGE

NEXT_PAGE ROUTINE             !DISPLAY NEXT PAGE
  SAVECNT# = RECORDS(TABLE) ! SAVE RECORD COUNT
  LOOP COUNT TIMES            ! FILL UP THE TABLE
    DO FILL_NEXT              !   FILL A TABLE ENTRY
    IF NOT FOUND              !   IF NONE ARE LEFT
      IF NOT SAVECNT#         !IF REBUILDING TABLE
        DO LAST_PAGE          !FILL IN RECORDS
        EXIT                  !EXIT OUT OF ROUTINE
      .
      BREAK                   !   EXIT LOOP
  . .
  DO SHOW_PAGE                ! DISPLAY THE PAGE

SET_NEXT ROUTINE              !POINT TO THE NEXT PAGE
  GET(TABLE,RECORDS(TABLE)) ! GET THE LAST TABLE ENTRY
  DO FILL_RECORD              ! FILL IN THE RECORD
  SET(DRU:BY_LASTNAME,DRU:BY_LASTNAME,TBL:PTR) ! POSITION FILE
  NEXT(DRUGDOG)               ! READ THE CURRENT RECORD

FILL_NEXT ROUTINE             !FILL NEXT TABLE ENTRY
```

```
  FOUND = FALSE              ! ASSUME RECORD NOT FOUND
  LOOP UNTIL EOF(DRUGDOG) !  LOOP UNTIL END OF FILE
    NEXT(DRUGDOG)            !   READ THE NEXT RECORD
    FOUND = TRUE             !   SET RECORD FOUND
    DO FILL_TABLE            !   FILL IN THE TABLE ENTRY
    ADD(TABLE)               !   ADD LAST TABLE ENTRY
    GET(TABLE,RECORDS(TABLE)-COUNT) !   GET ANY OVERFLOW RECORD
    DELETE(TABLE)            !   AND DELETE IT
    EXIT                     !   RETURN TO CALLER
  .
PREV_PAGE ROUTINE            !DISPLAY PREVIOUS PAGE
  LOOP COUNT TIMES           ! FILL UP THE TABLE
    DO FILL_PREV             !   FILL A TABLE ENTRY
    IF NOT FOUND THEN BREAK. !   GET OUT IF NO RECORD
  .
  DO SHOW_PAGE               ! DISPLAY THE PAGE

SET_PREV ROUTINE             !POINT TO PREVIOUS PAGE
  GET(TABLE,1)               ! GET THE FIRST TABLE ENTRY
  DO FILL_RECORD             ! FILL IN THE RECORD
  SET(DRU:BY_LASTNAME,DRU:BY_LASTNAME,TBL:PTR) !  POSITION FILE
  PREVIOUS(DRUGDOG)          ! READ THE CURRENT RECORD

FILL_PREV ROUTINE            !FILL PREVIOUS TABLE ENTRY
  FOUND = FALSE              ! ASSUME RECORD NOT FOUND
  LOOP UNTIL BOF(DRUGDOG) !  LOOP UNTIL BEGINNING OF FILE
    PREVIOUS(DRUGDOG)        !   READ THE PREVIOUS RECORD
    FOUND = TRUE             !   SET RECORD FOUND
    DO FILL_TABLE            !   FILL IN THE TABLE ENTRY
    ADD(TABLE,1)             !   ADD FIRST TABLE ENTRY
    GET(TABLE,COUNT+1)       !   GET ANY OVERFLOW RECORD
    DELETE(TABLE)            !   AND DELETE IT
    EXIT                     !   RETURN TO CALLER
  .
SHOW_PAGE ROUTINE            !DISPLAY THE PAGE
  NDX# = NDX                 ! SAVE SCREEN INDEX
  LOOP NDX = 1 TO RECORDS(TABLE) !  LOOP THRU THE TABLE
    GET(TABLE,NDX)           !   GET A TABLE ENTRY
    DO FILL_SCREEN           !   AND DISPLAY IT
    IF TBL:PTR = NEWPTR      !   SET INDEX FOR NEW RECORD
      NDX# = NDX             !   POINT TO CORRECT RECORD
  . .
  LOOP WHILE NDX <= COUNT !  FINISH BLANKING THE SCREEN
    BLANK(ROW(?POINT),COL(?POINT),ROWS(?POINT),COLS(?POINT))!BLANK A LINE
    NDX += 1                 !  INCREMENT NDX
  .
  NDX = NDX#                 ! RESTORE SCREEN INDEX
  NEWPTR = 0                 ! CLEAR NEW RECORD POINTER
  CLEAR(DRU:RECORD)          ! CLEAR RECORD AREA
```

```
FILL_TABLE ROUTINE        !MOVE FILE TO TABLE
  TBL:LASTNAME = DRU:LASTNAME
  TBL:FIRST_NAME = DRU:FIRST_NAME
  TBL:SSN = DRU:SSN
  TBL:RATE = DRU:RATE
  TBL:WORKCENT = DRU:WORKCENT
  TBL:PTR = POINTER(DRUGDOG) !  SAVE RECORD POINTER

FILL_RECORD ROUTINE      !MOVE TABLE TO FILE
  DRU:LASTNAME = TBL:LASTNAME

FILL_SCREEN ROUTINE      !MOVE TABLE TO SCREEN
  SCR:LASTNAME = TBL:LASTNAME
  SCR:FIRST_NAME = TBL:FIRST_NAME
  SCR:SSN = TBL:SSN
  SCR:RATE = TBL:RATE
  SCR:WORKCENT = TBL:WORKCENT

GET_RECORD ROUTINE       !GET SELECTED RECORD
  GET(TABLE,NDX)          !  GET TABLE ENTRY
  GET(DRUGDOG,TBL:PTR)  !  GET THE RECORD

*****************************************************************************

         MEMBER('DRUGDOG')
UPD_DRUGDOG  PROCEDURE

SCREEN       SCREEN        PRE(SCR),WINDOW(14,41),AT(7,20),HUE(15,4)
           ROW(1,40)  PAINT(1,2),TRN
           ROW(2,40)  PAINT(12,2),HUE(7,0),TRN
           ROW(14,1)  PAINT(1,2),TRN
           ROW(14,3)  PAINT(1,39),HUE(7,0),TRN
           ROW(1,1)     STRING('╔═{37}╗'),HUE(15,4)
           ROW(2,1)     REPEAT(11);STRING('‖<0{37}>‖'),HUE(15,4) .
           ROW(13,1)  STRING('╚═{37}╝'),HUE(15,4)
           ROW(2,13)  STRING('Update Personnel')
           ROW(5,4)     STRING('LASTNAME:')
           ROW(6,4)     STRING('FIRST NAME:')
           ROW(7,4)     STRING('SSN:')
             COL(8)     STRING('    '),HUE(7,4)
           ROW(8,4)     STRING('RATE:')
           ROW(9,4)     STRING('DEPT:')
           ROW(10,4)  STRING('DIVISION:')
           ROW(11,4)  STRING('WORKCENTER:')
MESSAGE      ROW(3,6)      STRING(30),HUE(15,4)
               ENTRY,USE(?FIRST_FIELD)
           ROW(5,16)  ENTRY(@s20),USE(DRU:LASTNAME),UPR,REQ,HUE(14,4)
                    SEL(14,0)
```

```
        ROW(6,16)  ENTRY(@s15),USE(DRU:FIRST_NAME),UPR,REQ,HUE(14,4)
|
                SEL(14,0)
        ROW(7,16) ENTRY(@N09),USE(DRU:SSN),HLP('SSN'),NUM,INS,HUE(14,4)|
                SEL(14,0)
        ROW(8,16)  ENTRY(@s6),USE(DRU:RATE),HLP('RATE'),UPR,HUE(14,4)   |
                SEL(14,0)
        ROW(9,16)  ENTRY(@s10),USE(DRU:DEPT),HLP('DEPT'),UPR,REQ        |
                HUE(14,4),SEL(14,0)
        ROW(10,16) ENTRY(@s10),USE(DRU:DIV),HLP('DIV'),UPR,REQ,HUE(14,4)|
                SEL(14,0)
        ROW(11,16) ENTRY(@s10),USE(DRU:WORKCENT),HLP('W/C'),UPR,REQ
|
                HUE(14,4),SEL(14,0)
               ENTRY,USE(?LAST_FIELD)
               PAUSE(''),USE(?DELETE_FIELD)
.

 EJECT
 CODE
 OPEN(SCREEN)              !OPEN THE SCREEN
 SETCURSOR                 !TURN OFF ANY CURSOR
 DISPLAY                   !DISPLAY THE FIELDS
 LOOP                      !LOOP THRU ALL THE FIELDS
  MEM:MESSAGE = CENTER(MEM:MESSAGE) !DISPLAY ACTION MESSAGE
  DO CALCFIELDS            !CALCULATE DISPLAY FIELDS
  ALERT                    !RESET ALERTED KEYS
  ALERT(ACCEPT_KEY)        !ALERT SCREEN ACCEPT KEY
  ALERT(REJECT_KEY)        !ALERT SCREEN REJECT KEY
  ACCEPT                   !READ A FIELD
  IF CHUCKED() = REJECT_KEY THEN RETURN. !RETURN ON SCREEN REJECT
KEY
  EXECUTE ACTION           !SET MESSAGE
    MEM:MESSAGE = 'Record will be Added' !
    MEM:MESSAGE = 'Record will be Changed' !
    MEM:MESSAGE = 'Press Enter to Delete' !
  .
  IF CHUCKED() = ACCEPT_KEY !ON SCREEN ACCEPT KEY
    UPDATE                 !  MOVE ALL FIELDS FROM SCREEN
    SELECT(?)              !  START WITH CURRENT FIELD
    SELECT                 !  EDIT ALL FIELDS
    CYCLE                  !  GO TO TOP OF LOOP
  .
  CASE FIELD()             !JUMP TO FIELD EDIT ROUTINE
  OF ?FIRST_FIELD          !FROM THE FIRST FIELD
    IF CHUCKED() = ESC_KEY THEN RETURN. !  RETURN ON ESC KEY
    IF ACTION = 3 THEN SELECT(?DELETE_FIELD).!   OR CONFIRM FOR DELETE

  OF ?DRU:SSN
```

```
    IF ~INRANGE(DRU:SSN,0,999999999) !IF FIELD IS OUT OF RANGE
      BEEP                  ! SOUND KEYBOARD ALARM
      SELECT(?DRU:SSN)      ! AND STAY ON THIS FIELD
      CYCLE                 ! GO TO TOP OF LOOP
    .

   OF ?LAST_FIELD           !FROM THE LAST FIELD
     IF CHUCKED() = 0 THEN SELECT(?). ! CHECK CHUCKED
    EXECUTE ACTION          ! UPDATE THE FILE
      ADD(DRUGDOG)          !   ADD NEW RECORD
      PUT(DRUGDOG)          !   CHANGE EXISTING RECORD
      DELETE(DRUGDOG)       !   DELETE EXISTING RECORD
    .
    IF ERRORCODE() = 40     ! DUPLICATE KEY ERROR
      MEM:MESSAGE = ERROR() !   DISPLAY ERR MESSAGE
      SELECT(2)             !   POSITION TO TOP OF FORM
      CYCLE                 !   GET OUT OF EDIT LOOP
    ELSIF ERROR()           ! CHECK FOR UNEXPECTED ERROR
      EXECUTE ACTION        !   BUILD AN ERROR MESSAGE
        ERROR" = 'Error: ' & ERROR() & ' ' | !FOR ADDING
              & 'Adding to DRUGDOG'
        ERROR" = 'Error: ' & ERROR() & ' ' | !FOR CHANGE
              & 'Changing DRUGDOG'
        ERROR" = 'Error: ' & ERROR() & ' ' | !FOR DELETING
              & 'Deleting from DRUGDOG'
      .
      STOP(ERROR")          !   HALT EXECUTION
    .
    ACTION = 0              ! SET ACTION TO COMPLETE
    RETURN                  ! AND RETURN TO CALLER

   OF ?DELETE_FIELD         !FROM THE DELETE FIELD
    IF CHUCKED() = ENTER_KEY | ! ON ENTER KEY
    OR CHUCKED() = ACCEPT_KEY ! OR CTRL-ENTER KEY
      SELECT(?LAST_FIELD) !   DELETE THE RECORD
    ELSE                    ! OTHERWISE
      BEEP                  !   BEEP AND ASK AGAIN
  . . .

CALCFIELDS   ROUTINE
  IF FIELD() > ?FIRST_FIELD !BEYOND FIRST_FIELD?
    IF CHUCKED() = 0 AND SELECTED() > FIELD() THEN EXIT. !GET OUT IF NOT
NONSTOP
  .
  SCR:MESSAGE = MEM:MESSAGE

*****************************************************************************
         MEMBER('DRUGDOG')
PERS_MENU    PROCEDURE
```

```
SCREEN       SCREEN
PRE(SCR),WINDOW(22,80),AT(1,1),HLP('PERS_MEN'),HUE(15,1)
          ROW(7,27)  PAINT(13,22),HUE(15,4)
          ROW(6,49)  PAINT(14,6),HUE(15,4)
          ROW(6,55)  PAINT(15,1),HUE(15,4)
          ROW(21,28) PAINT(1,29),HUE(15,0)
          ROW(7,57)  PAINT(15,2),HUE(15,0)
          ROW(1,1)     STRING('╔═{31}╦═{15}╦═{30}╗'),HUE(15,1)
          ROW(2,1)     STRING('‖<0{31}>‖<0{15}>‖<0{30}>‖'),HUE(15,1)
          ROW(3,1)     STRING('‖<0{31}>╚═{15}╝<0{30}>‖'),HUE(15,1)
          ROW(4,1)     REPEAT(19);STRING('‖<0{78}>‖'),HUE(15,1) .
          ROW(6,26)  STRING('╔═══════{21}═══════╗'),HUE(15,4)
          ROW(7,26)  STRING('‖<0{3}>│<0{21}>│<0{3}>‖'),HUE(15,4)
          ROW(8,26)  STRING('‖<0{3}>└─{21}┘<0{3}>‖'),HUE(15,4)
          ROW(9,26)  REPEAT(11);STRING('‖<0{29}>‖'),HUE(15,4) .
          ROW(20,26) STRING('╚═{29}╝'),HUE(15,4)
          ROW(2,34)  STRING(' DRUGDOG v 3.0 '),HUE(11,0)
          ROW(7,31)  STRING('R O S T E R   M E N U'),HUE(14,4)
                  ENTRY,USE(?FIRST_FIELD)
                  ENTRY,USE(?PRE_MENU)
                  MENU,USE(MENU_FIELD"),REQ
          ROW(10,32)   STRING('View All Personnel'),HUE(15,4),SEL(14,0)
          ROW(12,32)   STRING('View by Department'),HUE(15,4),SEL(14,0)
          ROW(14,33)   STRING('View by Division'),HUE(15,4),SEL(14,0)
          ROW(16,32)   STRING('View by Workcenter'),HUE(15,4),SEL(14,0)
          ROW(18,32)   STRING('Return to Main Menu'),HUE(15,4),SEL(14,0)
        .          .

 EJECT
 CODE
 OPEN(SCREEN)              !OPEN THE MENU SCREEN
 SETCURSOR                 !TURN OFF ANY CURSOR
 MENU_FIELD" = "           !START MENU WITH FIRST ITEM
 LOOP                      !LOOP UNTIL USER EXITS
  ALERT                    !TURN OFF ALL ALERTED KEYS
  ALERT(REJECT_KEY)        !ALERT SCREEN REJECT KEY
  ALERT(ACCEPT_KEY)        !ALERT SCREEN ACCEPT KEY
  ACCEPT                   !READ A FIELD OR MENU CHOICE
  IF CHUCKED() = REJECT_KEY THEN RETURN. !RETURN ON SCREEN REJECT

  IF CHUCKED() = ACCEPT_KEY !ON SCREEN ACCEPT KEY
   UPDATE                  !  MOVE ALL FIELDS FROM SCREEN
   SELECT(?)               !  START WITH CURRENT FIELD
   SELECT                  !  EDIT ALL FIELDS
   CYCLE                   !  GO TO TOP OF LOOP
  .                        !

  CASE FIELD()             !JUMP TO FIELD EDIT ROUTINE
  OF ?FIRST_FIELD          !FROM THE FIRST FIELD
```

```
    IF CHUCKED() = ESC_KEY THEN RETURN. !  RETURN ON ESC KEY

   OF ?PRE_MENU              !PRE MENU FIELD CONDITION
    IF CHUCKED() = ESC_KEY !  BACKING UP?
      SELECT(?-1)            !   SELECT PREVIOUS FIELD
    ELSE                     ! GOING FORWARD
      SELECT(?+1)            !   SELECT MENU FIELD
    .
   OF ?MENU_FIELD"           !FROM THE MENU FIELD
    EXECUTE CHOICE()         ! CALL THE SELECTED PROCEDURE
      SHO_LASTNAME           ! Show Drugdog By Lastname
      SHO_DEPT               ! Show Drugdog By Department
      SHO_DIV                ! Show Drugdog By Division
      SHO_WORKCENT           ! Show Drugdog By Workcenter
      RETURN
  . . .

*****************************************************************************

         MEMBER('DRUGDOG')
SHO_DEPT    PROCEDURE

SCREEN       SCREEN        PRE(SCR),WINDOW(25,80),AT(1,1),HUE(15,1)
           ROW(22,7)  PAINT(1,67),HUE(0,7)
           ROW(23,7)  PAINT(1,69),HUE(7,0)
           ROW(7,74)  PAINT(16,2),HUE(7,0)
           ROW(22,5)  PAINT(1,1),HUE(7,1)
           ROW(22,6)  PAINT(1,1),HUE(0,1)
           ROW(7,5)     PAINT(15,69),HUE(0,7)
           ROW(1,1)     STRING('╔═{22}╤═{33}╤═{21}╗'),HUE(15,1)
           ROW(2,1)     STRING('‖<0{22}>│<0{33}>│<0{21}>‖'),HUE(15,1)
           ROW(3,1)     STRING('‖<0{22}>└─{33}┘<0{21}>‖'),HUE(15,1)
           ROW(4,1)     REPEAT(4);STRING('‖<0{78}>‖'),HUE(15,1) .
           ROW(8,1)     STRING('‖<0{75},24,0,0>‖'),HUE(15,1)
           ROW(9,1)     STRING('‖<0,24,0{76}>‖'),HUE(15,1)
           ROW(10,1)  REPEAT(9);STRING('‖<0{78}>‖'),HUE(15,1) .
           ROW(19,1)  STRING('‖<0,25,0{76}>‖'),HUE(15,1)
           ROW(20,1)  STRING('‖<0{75},25,0,0>‖'),HUE(15,1)
           ROW(21,1)  REPEAT(2);STRING('‖<0{78}>‖'),HUE(15,1) .
           ROW(23,1)  STRING('‖├{78}┤‖'),HUE(15,1)
           ROW(24,1)  STRING('‖<0{78}>‖'),HUE(15,1)
           ROW(25,1)  STRING('╚═{78}╝'),HUE(15,1)
           ROW(22,7)  STRING('─{69}'),HUE(0,0)
           ROW(24,7)  STRING('=<16,0{13}>=<16,0{12}>=<16,0{14}>=<16>'      |
                    & '<0{15}>=<16>'),HUE(14,1)
           ROW(2,25)  STRING(' Command Personnel by Department '),HUE(11,0)
           ROW(4,30)  STRING('DEPARTMENT:')
           ROW(6,7)     STRING('LAST NAME {12}FIRST NAME {5}SSN {8}RATE')
             COL(62)  STRING('WORKCENTER'),HUE(15,1)
```

```
          ROW(9,77)  STRING('Pg')
          ROW(10,77) STRING('Up')
          ROW(18,77) STRING('Pg')
          ROW(19,77) STRING('Dn')
          ROW(24,4)  STRING('F1'),HUE(14,4)
            COL(9)       STRING(' Help'),HUE(14,1)
            COL(17)  STRING(' '),HUE(14,1)
            COL(18)  STRING('INS'),HUE(14,4)
            COL(25)  STRING('Add'),HUE(14,1)
            COL(32)  STRING('RTN'),HUE(14,4)
            COL(39)  STRING('Change   '),HUE(14,1)
            COL(48)  STRING('DEL'),HUE(14,4)
            COL(55)  STRING('Delete    '),HUE(14,1)
            COL(65)  STRING('ESC'),HUE(14,4)
            COL(72)  STRING('Exit'),HUE(14,1)
GET_DEPT      ROW(4,42)  STRING(10),HUE(14,1)
                ENTRY,USE(?FIRST_FIELD)
                ENTRY,USE(?PRE_POINT)
                REPEAT(15),EVERY(1),INDEX(NDX)
          ROW(7,5)       POINT(1,68),USE(?POINT),ESC(?-1)
LASTNAME      COL(6)         STRING(20)
FIRST_NAME    COL(27)    STRING(15)
SSN         COL(43)    STRING(@N09)
RATE          COL(54)    STRING(6)
WORKCENT      COL(62)    STRING(10)
          .          .

NDX     BYTE                !REPEAT INDEX FOR POINT AREA
ROW           BYTE          !ACTUAL ROW OF SCROLL AREA
COL     BYTE                !ACTUAL COLUMN OF SCROLL AREA
COUNT         BYTE(15)      !NUMBER OF ITEMS TO SCROLL
ROWS          BYTE(15)      !NUMBER OF ROWS TO SCROLL
COLS          BYTE(68)      !NUMBER OF COLUMNS TO SCROLL
FOUND         BYTE          !RECORD FOUND FLAG
NEWPTR        LONG          !POINTER TO NEW RECORD

TABLE         TABLE,PRE(TBL) !TABLE OF RECORD DATA
PTR       LONG                ! POINTER TO FILE RECORD
LASTNAME      STRING(20)
FIRST_NAME    STRING(15)
SSN       LONG
RATE          STRING(6)
WORKCENT      STRING(10)
DEPT          STRING(10)
          .

  EJECT
  CODE
  ACTION# = ACTION          !SAVE ACTION
```

```
  OPEN(SCREEN)              !OPEN THE SCREEN
  SETCURSOR                 !TURN OFF ANY CURSOR
  OBTAIN_DEPT               !CALL SETUP PROCEDURE
  TBL:PTR = 1               !START AT TABLE ENTRY
  NDX = 1                   !PUT SELECTOR BAR ON TOP ITEM
  ROW = ROW(?POINT)         !REMEMBER TOP ROW AND
  COL = COL(?POINT)         !LEFT COLUMN OF SCROLL AREA
  RECORDS# = TRUE           !INITIALIZE RECORDS FLAG
  CACHE(DRU:BY_DEPT,.25) !CACHE KEY FILE
  LOOP                      !LOOP UNTIL USER EXITS
   ACTION = ACTION#         !RESTORE ACTION
   SCR:GET_DEPT = MEM:GET_DEPT
   ALERT                    !RESET ALERTED KEYS
   ALERT(REJECT_KEY)        !ALERT SCREEN REJECT KEY
   ALERT(ACCEPT_KEY)        !ALERT SCREEN ACCEPT KEY
   ACCEPT                   !READ A FIELD
   IF CHUCKED() = REJECT_KEY THEN BREAK. !RETURN  ON  SCREEN  REJECT
KEY
   IF CHUCKED() = ACCEPT_KEY  ! !ON SCREEN ACCEPT KEY
   AND FIELD() <> ?POINT   !BUT NOT ON THE POINT FIELD
    UPDATE                  !  MOVE ALL FIELDS FROM SCREEN
    SELECT(?)               !  START WITH CURRENT FIELD
    SELECT                  !  EDIT ALL FIELDS
    CYCLE                   !  GO TO TOP OF LOOP
   .

   CASE FIELD()             !JUMP TO FIELD EDIT ROUTINE

   OF ?FIRST_FIELD          !FROM THE FIRST FIELD
    IF CHUCKED() = ESC_KEY   ! !  RETURN ON ESC KEY
    OR RECORDS# = FALSE !  OR NO RECORDS
     BREAK                  !    EXIT PROCEDURE
    .
    IF ACTION = 4           !  TABLE LOOKUP REQUEST
     NEWPTR = POINTER(DRUGDOG) !  SET POINTER TO RECORD
     IF NOT NEWPTR          !  RECORD NOT PASSED TO TABLE
      SET(DRU:BY_DEPT,DRU:BY_DEPT) !    POSITION TO CLOSEST RECORD
      NEXT(DRUGDOG)         !    READ RECORD
      NEWPTR = POINTER(DRUGDOG) !    SET POINTER
     .
     DO FIND_RECORD         !  POSITION FILE
    ELSE
     NDX = 1                !  PUT SELECTOR BAR ON TOP ITEM
     DO FIRST_PAGE          !  BUILD MEMORY TABLE OF KEYS
    .
    RECORDS# = TRUE         !  ASSUME THERE ARE RECORDS
   OF ?PRE_POINT            !PRE POINT FIELD CONDITION
    IF CHUCKED() = ESC_KEY !  BACKING UP?
     SELECT(?-1)            !    SELECT PREVIOUS FIELD
```

```
    ELSE                    ! GOING FORWARD
      SELECT(?POINT)        !   SELECT MENU FIELD
    .
   OF ?POINT                !PROCESS THE POINT FIELD
    IF RECORDS(TABLE) = 0 !IF THERE ARE NO RECORDS
      CLEAR(DRU:RECORD) !  CLEAR RECORD AREA
      ACTION = 1            ! SET ACTION TO ADD
      GET(DRUGDOG,0)        ! CLEAR PENDING RECORD
      UPD_DRUGDOG           ! CALL FORM FOR NEW RECORD
      NEWPTR = POINTER(DRUGDOG) !    SET POINTER TO NEW RECORD
      DO FIRST_PAGE         ! DISPLAY THE FIRST PAGE
      IF RECORDS(TABLE) = 0 !  IF THERE AREN'T ANY RECORDS
        RECORDS# = FALSE    !   INDICATE NO RECORDS
        SELECT(?PRE_POINT-1) !    SELECT THE PRIOR FIELD
      .
      CYCLE                 !   AND LOOP AGAIN
    .
    CASE CHUCKED()          !PROCESS THE KEYSTROKE

    OF INS_KEY              !INS KEY
      CLEAR(DRU:RECORD) !  CLEAR RECORD AREA
      ACTION = 1            ! SET ACTION TO ADD
      GET(DRUGDOG,0)        ! CLEAR PENDING RECORD
      UPD_DRUGDOG           ! CALL FORM FOR NEW RECORD
      IF ~ACTION            ! IF RECORD WAS ADDED
        NEWPTR = POINTER(DRUGDOG) !    SET POINTER TO NEW RECORD
        DO FIND_RECORD      !   POSITION IN FILE
      .
    OF ENTER_KEY            !ENTER KEY
    OROF ACCEPT_KEY         !CTRL-ENTER KEY
      DO GET_RECORD         ! GET THE SELECTED RECORD
      IF ACTION = 4 AND CHUCKED() = ENTER_KEY!     IF THIS IS A LOOKUP
REQUEST
        ACTION = 0          !   SET ACTION TO COMPLETE
        BREAK               !   AND RETURN TO CALLER
      .
      IF ~ERROR()           ! IF RECORD IS STILL THERE
        ACTION = 2          !   SET ACTION TO CHANGE
        UPD_DRUGDOG         !   CALL FORM TO CHANGE REC
        IF ACTION           !   IF SUCCESSFUL RE-DISPLAY
          ACTN# = 0 THEN CYCLE.
      .
      NEWPTR = POINTER(DRUGDOG) !    SET POINTER TO NEW RECORD
      DO FIND_RECORD        !   POSITION IN FILE
    OF DEL_KEY              !DEL KEY
      DO GET_RECORD         ! READ THE SELECTED RECORD
      IF ~ERROR()           ! IF RECORD IS STILL THERE
        ACTION = 3          !   SET ACTION TO DELETE
        UPD_DRUGDOG         !   CALL FORM TO DELETE
```

```
      IF ~ACTION              !   IF SUCCESSFUL
        N# = NDX              !SAVE POINT INDEX
        DO SAME_PAGE          !RE-DISPLAY
        NDX = N#              !RESTORE POINT INDEX
    . .
   OF DOWN_KEY                !DOWN ARROW KEY
     DO SET_NEXT              ! POINT TO NEXT RECORD
     DO FILL_NEXT             ! FILL A TABLE ENTRY
     IF FOUND                 ! FOUND A NEW RECORD
       SCROLL(ROW,COL,ROWS,COLS,ROWS(?POINT))! SCROLL THE SCREEN UP
       GET(TABLE,RECORDS(TABLE)) ! GET RECORD FROM TABLE
       DO FILL_SCREEN         ! DISPLAY ON SCREEN
     .

   OF PGDN_KEY                !PAGE DOWN KEY
     DO SET_NEXT              ! POINT TO NEXT RECORD
     DO NEXT_PAGE             ! DISPLAY THE NEXT PAGE

   OF CTRL_PGDN               !CTRL-PAGE DOWN KEY
     DO LAST_PAGE             ! DISPLAY THE LAST PAGE
     NDX = RECORDS(TABLE) !  POSITION POINT BAR

   OF UP_KEY                  !UP ARROW KEY
     DO SET_PREV              ! POINT TO PREVIOUS RECORD
     DO FILL_PREV             ! FILL A TABLE ENTRY
     IF FOUND                 ! FOUND A NEW RECORD
       SCROLL(ROW,COL,ROWS,COLS,-(ROWS(?POINT)))! SCROLL THE SCREEN
DOWN
       GET(TABLE,1)           ! GET RECORD FROM TABLE
       DO FILL_SCREEN         ! DISPLAY ON SCREEN
     .

   OF PGUP_KEY                !PAGE UP KEY
     DO SET_PREV              ! POINT TO PREVIOUS RECORD
     DO PREV_PAGE             ! DISPLAY THE PREVIOUS PAGE

   OF CTRL_PGUP               !CTRL-PAGE UP
     DO FIRST_PAGE            ! DISPLAY THE FIRST PAGE
     NDX = 1                  ! POSITION POINT BAR
 . . .
 FREE(TABLE)                  !FREE MEMORY TABLE
 FREE(DRU:BY_DEPT)            !FREE CACHE
 RETURN                       !AND RETURN TO CALLER

SAME_PAGE ROUTINE             !DISPLAY THE SAME PAGE
 GET(TABLE,1)                 ! GET THE FIRST TABLE ENTRY
 DO FILL_RECORD               ! FILL IN THE RECORD
 SET(DRU:BY_DEPT,DRU:BY_DEPT,TBL:PTR) ! POSITION FILE
 FREE(TABLE)                  ! EMPTY THE TABLE
```

```
  DO NEXT_PAGE              ! DISPLAY A FULL PAGE

FIRST_PAGE ROUTINE        !DISPLAY FIRST PAGE
  FREE(TABLE)               ! EMPTY THE TABLE
  CLEAR(DRU:RECORD,-1)      ! CLEAR RECORD TO LOW VALUES
  CLEAR(TBL:PTR)            ! ZERO RECORD POINTER
  SET(DRU:BY_DEPT)          ! POINT TO FIRST RECORD
  LOOP NDX = 1 TO COUNT !  FILL UP THE TABLE
    DO FILL_NEXT            !   FILL A TABLE ENTRY
    IF NOT FOUND THEN BREAK. !   GET OUT IF NO RECORD
  .
  NDX = 1                   ! SET TO TOP OF TABLE
  DO SHOW_PAGE              ! DISPLAY THE PAGE

LAST_PAGE ROUTINE         !DISPLAY LAST PAGE
  NDX# = NDX                ! SAVE SELECTOR POSITION
  FREE(TABLE)               ! EMPTY THE TABLE
  CLEAR(DRU:RECORD,1)       ! CLEAR RECORD TO HIGH VALUES
  CLEAR(TBL:PTR,1)          ! CLEAR PTR TO HIGH VALUE
  SET(DRU:BY_DEPT)          ! POINT TO FIRST RECORD
  LOOP NDX = COUNT TO 1 BY -1 !  FILL UP THE TABLE
    DO FILL_PREV            !   FILL A TABLE ENTRY
    IF NOT FOUND THEN BREAK. !   GET OUT IF NO RECORD
  .                         ! END OF LOOP
  NDX = NDX#                ! RESTORE SELECTOR POSITION
  DO SHOW_PAGE              ! DISPLAY THE PAGE

FIND_RECORD ROUTINE     !POSITION TO SPECIFIC RECORD
  SET(DRU:BY_DEPT,DRU:BY_DEPT,NEWPTR) !POSITION FILE
  IF NEWPTR = 0             !NEWPTR NOT SET
    NEXT(DRUGDOG)           ! READ NEXT RECORD
    NEWPTR = POINTER(DRUGDOG) !  SET NEWPTR
    SKIP(DRUGDOG,-1)        ! BACK UP TO DISPLAY RECORD
  .
  FREE(TABLE)               ! CLEAR THE RECORD
  DO NEXT_PAGE              ! DISPLAY A PAGE

NEXT_PAGE ROUTINE         !DISPLAY NEXT PAGE
  SAVECNT# = RECORDS(TABLE) !  SAVE RECORD COUNT
  LOOP COUNT TIMES          ! FILL UP THE TABLE
    DO FILL_NEXT            !   FILL A TABLE ENTRY
    IF NOT FOUND            !   IF NONE ARE LEFT
      IF NOT SAVECNT#       !IF REBUILDING TABLE
        DO LAST_PAGE        !FILL IN RECORDS
        EXIT                !EXIT OUT OF ROUTINE
      .
      BREAK                 !   EXIT LOOP
  . .
  DO SHOW_PAGE              ! DISPLAY THE PAGE
```

```
SET_NEXT ROUTINE          !POINT TO THE NEXT PAGE
  GET(TABLE,RECORDS(TABLE)) !  GET THE LAST TABLE ENTRY
  DO FILL_RECORD          !  FILL IN THE RECORD
  SET(DRU:BY_DEPT,DRU:BY_DEPT,TBL:PTR) !  POSITION FILE
  NEXT(DRUGDOG)           !  READ THE CURRENT RECORD

FILL_NEXT ROUTINE         !FILL NEXT TABLE ENTRY
  FOUND = FALSE           !  ASSUME RECORD NOT FOUND
  LOOP UNTIL EOF(DRUGDOG) !  LOOP UNTIL END OF FILE
    NEXT(DRUGDOG)         !    READ THE NEXT RECORD
    IF ~(DRU:DEPT = MEM:GET_DEPT) THEN CYCLE. !    FILTER
    FOUND = TRUE          !    SET RECORD FOUND
    DO FILL_TABLE         !    FILL IN THE TABLE ENTRY
    ADD(TABLE)            !    ADD LAST TABLE ENTRY
    GET(TABLE,RECORDS(TABLE)-COUNT) !    GET ANY OVERFLOW RECORD
    DELETE(TABLE)         !    AND DELETE IT
    EXIT                  !    RETURN TO CALLER
  .
PREV_PAGE ROUTINE         !DISPLAY PREVIOUS PAGE
  LOOP COUNT TIMES        !  FILL UP THE TABLE
    DO FILL_PREV          !    FILL A TABLE ENTRY
    IF NOT FOUND THEN BREAK. !    GET OUT IF NO RECORD
  .
  DO SHOW_PAGE            !  DISPLAY THE PAGE

SET_PREV ROUTINE          !POINT TO PREVIOUS PAGE
  GET(TABLE,1)            !  GET THE FIRST TABLE ENTRY
  DO FILL_RECORD          !  FILL IN THE RECORD
  SET(DRU:BY_DEPT,DRU:BY_DEPT,TBL:PTR) !  POSITION FILE
  PREVIOUS(DRUGDOG)       !  READ THE CURRENT RECORD

FILL_PREV ROUTINE         !FILL PREVIOUS TABLE ENTRY
  FOUND = FALSE           !  ASSUME RECORD NOT FOUND
  LOOP UNTIL BOF(DRUGDOG) !  LOOP UNTIL BEGINNING OF FILE
    PREVIOUS(DRUGDOG)     !    READ THE PREVIOUS RECORD
    IF ~(DRU:DEPT = MEM:GET_DEPT) THEN CYCLE. !    FILTER
    FOUND = TRUE          !    SET RECORD FOUND
    DO FILL_TABLE         !    FILL IN THE TABLE ENTRY
    ADD(TABLE,1)          !    ADD FIRST TABLE ENTRY
    GET(TABLE,COUNT+1)    !    GET ANY OVERFLOW RECORD
    DELETE(TABLE)         !    AND DELETE IT
    EXIT                  !    RETURN TO CALLER
  .
SHOW_PAGE ROUTINE         !DISPLAY THE PAGE
  NDX# = NDX              !  SAVE SCREEN INDEX
  LOOP NDX = 1 TO RECORDS(TABLE) !  LOOP THRU THE TABLE
    GET(TABLE,NDX)        !    GET A TABLE ENTRY
    DO FILL_SCREEN        !    AND DISPLAY IT
    IF TBL:PTR = NEWPTR   !    SET INDEX FOR NEW RECORD
```

```
    NDX# = NDX                  !   POINT TO CORRECT RECORD
  . .
  LOOP WHILE NDX <= COUNT !  FINISH BLANKING THE SCREEN
    BLANK(ROW(?POINT),COL(?POINT),ROWS(?POINT),COLS(?POINT))!BLANK A LINE
    NDX += 1                    !   INCREMENT NDX
  .
  NDX = NDX#                    !  RESTORE SCREEN INDEX
  NEWPTR = 0                    !  CLEAR NEW RECORD POINTER
  CLEAR(DRU:RECORD)             !  CLEAR RECORD AREA

FILL_TABLE ROUTINE              !MOVE FILE TO TABLE
  TBL:LASTNAME = DRU:LASTNAME
  TBL:FIRST_NAME = DRU:FIRST_NAME
  TBL:SSN = DRU:SSN
  TBL:RATE = DRU:RATE
  TBL:WORKCENT = DRU:WORKCENT
  TBL:DEPT = DRU:DEPT
  TBL:PTR = POINTER(DRUGDOG) !  SAVE RECORD POINTER

FILL_RECORD ROUTINE             !MOVE TABLE TO FILE
  DRU:DEPT = TBL:DEPT
  DRU:LASTNAME = TBL:LASTNAME

FILL_SCREEN ROUTINE             !MOVE TABLE TO SCREEN
  SCR:LASTNAME = TBL:LASTNAME
  SCR:FIRST_NAME = TBL:FIRST_NAME
  SCR:SSN = TBL:SSN
  SCR:RATE = TBL:RATE
  SCR:WORKCENT = TBL:WORKCENT

GET_RECORD ROUTINE              !GET SELECTED RECORD
  GET(TABLE,NDX)                !  GET TABLE ENTRY
  GET(DRUGDOG,TBL:PTR)          !  GET THE RECORD
```

********** END OF *DRUGDOG* 3.0 SOURCE CODE **********

APPENDIX D

DRUGDOG 3.0 USER'S MANUAL

DRUGDOG 3.0 Users' Manual by D. E. Wilson, LT, USN

SECTION

INTRODUCTION

1.1 System Overview
1.2 Software Contents

INSTALLATION

2.1 Computer Requirements
2.2 DOS and Printer Requirements
2.3 Hard Drive Considerations
2.4 DDINSTAL.EXE Installation Software
2.5 Floppy Disk Installation

SOFTWARE SPECIFICS

3.0 DRUGDOG MAIN MENU
3.1 Conduct Random Urinalysis
3.2 Roster Maintenance
3.3 Print Reports
3.4 Database Utilities
3.5 Users Guide

4.0 ROSTER MENU
4.1 View All Personnel
4.2 Adding a new record
4.3 Changing a record
4.4 Deleting a Record
4.5 Other Table Views

DRUGDOG 3.0 Users' Manual by D. E. Wilson, LT, USN

DRUGDOG Overview

The DRUGDOG Software Package was designed to assist commands in managing their Random Urinalysis Program. It provides the Command Urinalysis Coordinator a device to painlessly select a truly random list of personnel to undergo testing. Generally, a "command quota" must be met on a monthly basis, and the selection process must be completely random in nature. Written in the Clarion fourth generation language, DRUGDOG is exceptionally easy to use. Unlike dBase, Enable or other complicated database programs, there are no difficult commands to learn, no thick reference manuals to decipher. A clear, simple menu format drives all program functions. Even those with no prior computer experience can master the program in minimal time.

How DRUGDOG Works:

The user builds a file called "DRUGDOG.DAT" that is essentially a command "Alpha" roster containing all personnel in the command. To generate a Random Urinalysis listing, merely tell DRUGDOG your "command quota" number, and within seconds, DRUGDOG generates the random numbers, retrieves the corresponding personnel from the roster file, sorts them in alphabetical order and displays them on-screen. The listing may be printed in Department, Division, or WorkCenter format.

Roster Maintenance:

DRUGDOG provides easy functions to delete personnel from the roster who have transferred, add newly reported personnel, update the rank of promoted personnel or change the last name of personnel due to marriage. One may check to see if an individual is in the roster by a unique "quick-find" locator. The entire roster may be reviewed online or printed out in alphabetic order to ensure accuracy and completeness.

1.2 Software Contents

As with any new software, be sure to make working copy of DRUGDOG and put the original distribution disk away in a safe place! Although great effort was spent making DRUGDOG as user-friendly as possible, there is NO substitute for making backup copies of your working disk EVERY time you update the roster file.

A. DRUGDOG.EXE
Executive program used to control all primary functions in this software package. Menu driven format.

B. DRUGDOG.DAT
Random data file containing a list of all command personnel. Each personnel record requires only 81 bytes of RAM.

C. MANUAL.EXE
Allows the user to read the Users' Manual directly from the computer's monitor. (Uses MANUAL file)

D. MANUAL
The file containing this Users' Manual. A print-out can be made and retained for future reference. To print the manual, insert disk into drive A, ensure you printer is on, and paper is ready. Note: Ensur the print head is positioned at the top of a new pa The file will automatically advance a new page as needed.

E. DDINSTAL.EXE
Installation program to place the DRUGDOG package on hard drive C. However, be sure to read Section 2.3, "Hard Drive Considerations" prior to running this file.

F. DRUGDOG.HLP
Context-sensitive online help file

G. DRUGDOG.K##
Key files for database processing.

2.0 INTRODUCTION

2.1 Hardware Requirements

The DRUGDOG software package is compatible with the IBM PC-XT, AT, and PS/2 machines and on all clones claiming compatibility. The INTEL 80x86 or higher Central Processing Unit (CPU) is required to ensure proper execution of this software. A EGA/VGA color video driver will ensure full video compatibility.

A minimum of one floppy disk drive and 360K of RAM is needed to effectively utilize this software. This software has been tested on machines with an internal clock rate from 4.77 Mhz to 66 Mhz. DRUGDOG may be installed on the hard disk for optimum performance, but be sure to read section 2-3, "Hard Disk Considerations" for issues concerning overall security and the Privacy Act.

2.2 DOS & Printer Requirements

DOS version 3.0 or later should be used for correct performance. Zenith systems provide the correct version of DOS. Memory-resident (TSR) software such as SideKick or AutoMenu programs reduce the amount of available RAM, which may result in unfortunate error messages.

Any IBM compatible dot matrix printer with tractor paper feed will adequately provide print-outs of both the Master Roster and individual Random Urinalysis listings. Alps, Okidata, Citizen, Epson and Panasonic are a few of many compatible printers. Primmages printers with continuous single sheet feeders will also work.

2.3 Hard Disk Considerations

There are two issues to consider when installing this application to the main hard drive:

a) The data used by this program (the Social Security Numbers) is covered by the Privacy Act. For this reason you must be sure to lock up the working disk when not in use and treat the disk as you normally treat classified magnetic media under your command's ADP Security Instructions.

b) Additionally, the very nature of this program, i.e. selecting personnel for random urinalysis, makes it more likely to be tampere with. Indeed, if you discover, upon attempting to review your roster, that nothing but machine-language "garbage" comes up on screen, it's possible that someone has corrupted your file by attempting to remove their record from the list. Readily common utility programs (ex:PC Tools or Norton Commander) give even non-hackers the ability to edit data files, sometimes with catastrophic results.

For these reasons, you may want to consider running DRUGDOG entirely from floppy disk. There will be a noticeable increase in time required to complete processing due to the difference in disk access times. Generally, database programs are very disk-intensive, so floppy drive operation is not recommended.

2.4 "DDINSTAL" Custom Hard Disk Installation Software

If you do have a secure ADP system, DO install DRUGDOG on the hard drive. This is the RECOMMENDED method when security allows

a) Turn on your computer and insert your copy of DRUGDOG into a floppy drive. This disk MUST HAVE all the files listed in sectior 1.2 above, in addition to DDINSTAL.EXE. At the prompt, enter:

DDINSTAL.EXE

b) The install program will ask which floppy drive, A or B, the installation disk is in. It will then create a sub-directory called C:\DRUGDOG, and all files will be copied into it. On-screen message show the progress of each file during installation, and also when the installation is complete.

To run DRUGDOG, go to your new "C:\DRUGDOG" directory and enter: DRUGDOG.EXE

c) "Power users" who insist on (or are spoiled by) speed and therefore MUST run software exclusively on the hard drive, can still have a degree of security if they remove the "DRUGDOG.DAT" file from the C:\DRUGDOG sub-directory after each use, after performing a backup with it to a floppy disk. Again, as always, one should always make backups after updating the roster.

2.5 FLOPPY DISK INSTALLATION

First and foremost, MAKE A BACKUP COPY OF YOUR ORIGINAL DISK!! The importance of this cannot be overemphasize! Use your favorite utility program, or the following DOS procedure: at the DOS prompt, type in:

DISKCOPY A: (then press <RTN>)

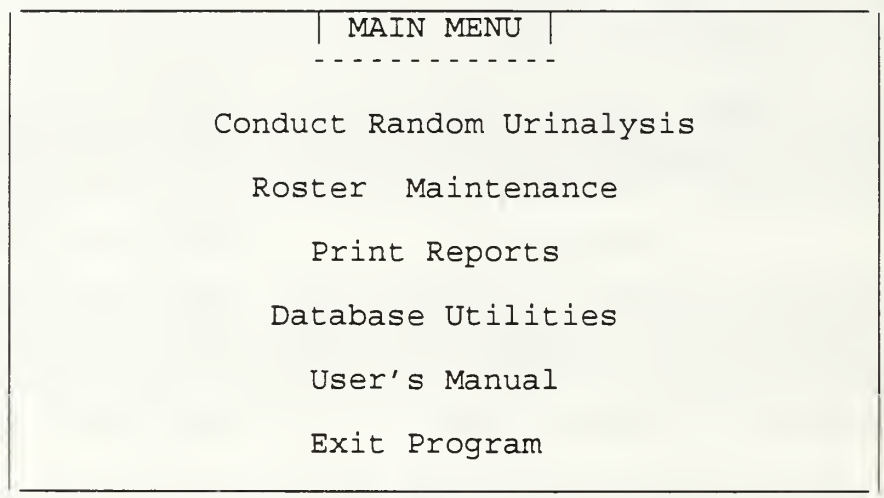
When completed, use your new working copy. Put your original disk away in a safe place. Whenever you need to use the program, insert your working disk into drive "A" and enter

DRUGDOG.EXE (then press <RTN>)

The program takes a few seconds to load up from floppy. The next few sections describe the operation and different features found in DRUGDOG.

3.0 DRUGDOG MAIN MENU

After opening its data files, the program Main Menu will come up on screen. Of course, the actual Main Menu is much more intense than the below "stick-picture"" menu. The menu items are the focus of this section.



From this Main Menu, all program functions are executed. You are returned to this menu upon exiting the selected function. To select a menu item, place the highlight bar on that item and press <RTN>.

With the exception of "Exit Program", these Main Menu items are described in sections 3.1 - 3.5. These functions are fairly self-explanatory. The user interface was carefully designed to give the user the intuitive feel of most widely available commercial software.

For example, as one navigates through the menu hierarchy, it becomes apparent that, as a general rule, the <RTN> key usually gets you the next layer "down" into the program. Conversely, pressing <ESC> usually allows you to back out a step just before you get into trouble. This seems true throughout the application.

Still, most users want an extensive on-line help system, available at any point by just pressing the "F1" key. This help system is context-sensitive, so that help is displayed on that screen, window, form or data entry field, etc. you are on when F1 is pressed. These help screens are comprehensive enough that it makes referencing a user's manual almost unnecessary.

3.1 CONDUCT RANDOM URINALYSIS

It is appropriate for this to be the first choice in the main menu hierarchy. The focus of this application is to create a random list of personnel. This is where the rubber meets the road. When selected, a serious dialogue box pops up and DRUGDOG displays 4 numeric choices that represent 5% to 20% of the command.

The user enters the total number of personnel required for this test period (the command "quota") based on a current database, After entering the "quota" number, pressing <RTN> generates the list and displays it on screen. All names will be in alphabetical order.

3.2 ROSTER MAINTENANCE

This item will display a second menu that provides all facilities needed for adding, deleting or changing personnel data. It is the user's responsibility to ensure this roster is current prior to generating a random listing.

One can browse personnel records through four different table listings: by Department, by Division, by WorkCenter, or the entire command, all alphabetically by last name.

Each option is fully explained in section 4.0

3.3 PRINT REPORTS

This item pops up a second menu that offers various print reports. Typically, those individuals just selected for urinalysis can be printed for distribution. The list can be filtered by Department, Division, or Workcenter. One can also retrieve and print out past urinalysis listings from historical records.

3.4 DATABASE UTILITIES

Two utilities are provided to keep DRUGDOG operating at peak performance and to safeguard your data: Backups and Re-Indexing.

A) BACKUP TO FLOPPY:

Insert a formatted disk in drive A and press <RTN>. A complete copy of DRUGDOG.DAT is placed on this floppy. A dialogue box indicates when copying is completed. This can save many hours of agony if the data file is ever lost or destroyed on the hard drive.

B) RE-INDEX DATA FILES:

As the database is used over a period of time, the constant rearrangement, additions and deletions of records in the data file can lead to a noticeable decrease in system performance. Re-indexing is a method to "shuffle" the database deck and return the file into its optimal condition.

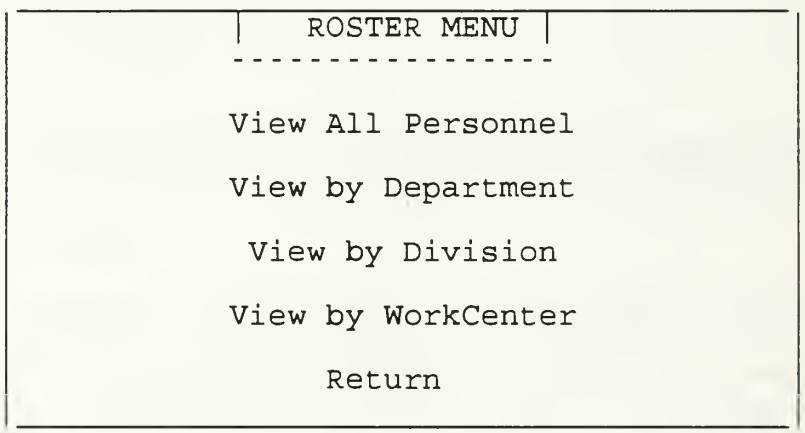
Ensure you run this utility on a routine basis. Re-Indexing takes just a few minutes, and can make a tremendous difference in processing time to those who must execute DRUGDOG on floppy disks!

3.5 USER'S MANUAL

If you are reading this at the computer monitor, you already know what this menu item is and how it works. It displays this manual, allows fast top-to-bottom scrolling, and offers to print out this manual for later enjoyment. Before printing, ensure that the printer head is at the start of a new page. The file will automatically form-feed the paper as necessary.

4.0 ROSTER MENU

Selecting "Roster Maintenance" from the Main Menu will pop up a second menu shown below. This allows one to access records in a few different ways. In each case, a table is displayed showing an alphabetical list of personnel. The table has a scrolling highlight bar which is used to retrieve these personnel records for view or modification. This menu looks as follows:



All functions are fairly self-explanatory and are discussed in the following four sections.

As with most standard database-oriented programs, there are three primary actions involved in maintaining your roster: record additions, record deletions, and record modifications. This software was designed so that only one simple form handles all of these actions.

4.1 VIEW BY PERSONNEL

Selecting this item from the Personnel Roster Menu will display the entire command in alphabetical order. Note at the top of the table a unique "locator" feature. Instead of scrolling down through several hundred records to find "WILLIAMS", the user just enters the first few letters of the member's last name. In this case, one would enter "WIL" in this locator field.

As soon as entries are made, the highlight bar automatically travels to the first record that matches what is entered in the locator thus far. This usually places the highlight bar very close (and often, on top of) the desired record.

This table is also your entrance to the heart of database activities: record maintenance. The bottom of the view table list everything you need to do at this table. If the highlight bar is resting on a record, expect action if the insert, delete, or Return key is pressed. There's always F1 help, if needed.

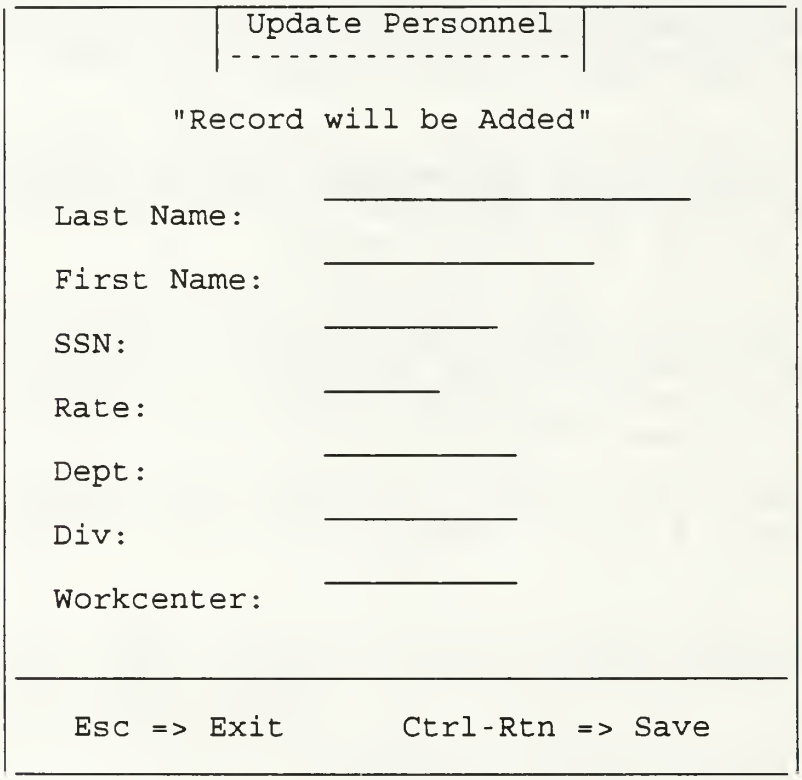
If the data file is empty or missing upon program start, DRUGDOG will create a new (but empty) data file, including the key files.

The next three sections describe how to add, delete, and modify data records directly from the display table itself. Additionally, there is only one standardized input form, which serves all three record transactions, saving users from having to learn three different transaction forms.

4.1.1 Adding a new record

To add new personnel (records) to the database, just select any of the "View..." roster menu items. The resulting roster table lists all members in alphabetical order. Press the "Insert" key. Instantly, a "bright white on red" data record input form pops up, casting a shadow on the table underneath.

The cursor is set at the first data field "Last Name". This is the "active" data field for editing. Not only is the cursor at the start of the field an indication, but the field itself changes colors to a bright yellow on black. Below is an example of a blank record input form:



One can move between data fields using the keyboard arrows. Once a field has been completed, pressing <RTN> highlights the next field down, and the cycle repeats. When the last field is filled, the record is saved, the input form disappears, and the record table is restored as before. However, the new record just saved to disk is also displayed.

4.1.2 Modifying a record

Once the record is found by the "quick-find" locator or just scrolling to it, making a change to a record is easy. Place the highlight bar over the record, and press <RTN>. The identical record form pops up as before, it's data fields filled with the existing data on this record. Below is an example of the "WILLIAMS" record:

```
                Update Personnel
                ----------------

            "Record will be Changed"

  Last Name:      WILLIAMS

  First Name:     JONATHAN

  SSN:            3382949330

  Rate:           AD1

  Dept:           MAINTENANCE

  Div:            AIRCRAFT

  Workcenter:     POWERPLANTS

   Esc => Exit        Ctrl-Rtn => Save
```

The cursor is set at the first data field "Last Name". Again, you can use the arrow keys to jump right to the field you want to modify and then enter the changes. If only one or two fields were changed, press <Ctrl-Rtn> together, and the record is saved. Pressing <RTN> through all the unchanged fields also saves the record changes. Pressing <Esc> collapses the input form and does not retain any changes made to it before escaping.

4.1.3 Record Field Entries

DRUGDOG's record field entries are error-trapped to assist new users and aid in maintaining database integrity. The record input form below shows, via a legend, some of the error-trapping algorithms that were encoded. This prevents, for example, entering letters in a Zip Code entry field, or trying to enter a Zip Code less than nine digits in length.

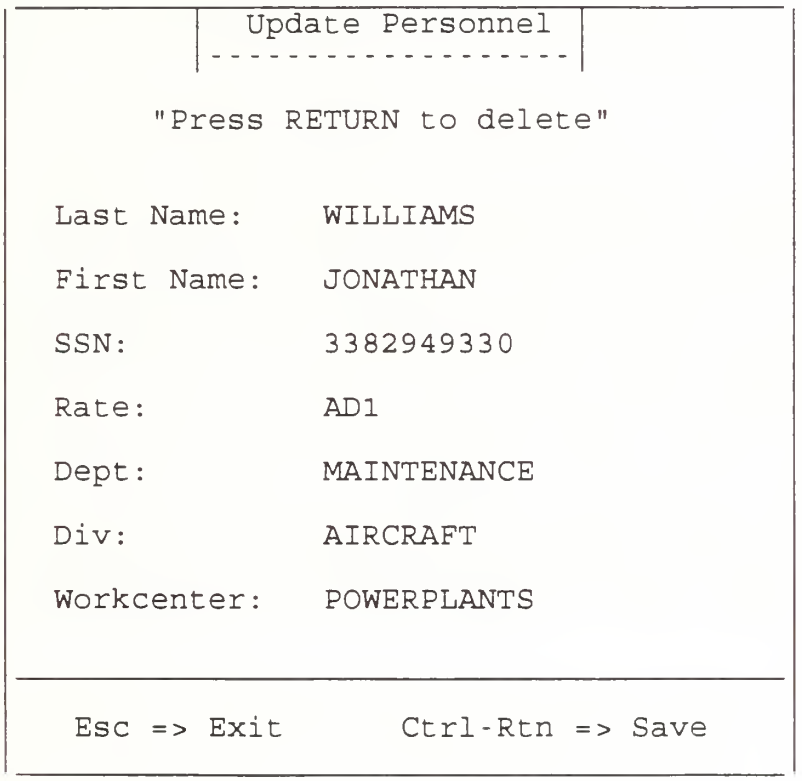
```
                Update Personnel

      "Press RETURN to delete"

  Last Name:     {REQ, UPP}
  First Name:    {REQ, UPP}
  SSN:           {REQ, IMM, 9NUM}
  Rate:          {UPP}
  Dept:          {UPP, CON}
  Div:           {UPP, REQ, CON}
  Workcenter:    {UPP, REQ, CON}

   Esc => Exit       Ctrl-Rtn => Save
```

```
-= Legend =-

REQ - Required data entry

UPP - Upper case letters
      are automatic

IMM - Immediate entry: as
      soon as the last
      digit is entered,
      entry field moves c

9NUM - a Zip Code must be
       9 numeric digits
       between 000000000
       999999999

CON  - All codes or names
       MUST be Consistent
       See section 4.2
```

Use <RTN> or arrow keys to move the entry field as needed. The message at the top of the form will always indicate the current record transaction type. Using F1 Help provides a detailed page of field-by-field info on what you can and can't do during record entry.

4.1.4 Deleting a record

Everybody loves to delete records, right? No problem here! Highlight the desired record as you would normally, but press the <Del> key for deletion. The now familiar record input form pops up with the record and asks the user to press <RTN> to confirm the deletion. Careful... pressing <RTN> too fast can cost valuable data loss and promote bad attitudes.



Above, the "WILLIAMS" record is finally about to be history. Note the message "Press RETURN to Delete". Only the <RTN> key will actually delete the record; most any other key will not. If you erred and do not want to delete the highlighted record, the <ESC> key will always bail you out.

4.2 VIEW BY DEPARTMENT / DIVISION / WORKCENTER

These modules look, operate, and behave exactly as the tables from "View All Personnel" menu item, with one exception: they take the concept of the "quick-find" locator idea to a higher level.

In selecting one of these three tables, the first task is to be confronted with telling the system exactly which Department, Division, WorkCenter is desired for "filtering". When the table is assembled, it will only pull those records who match the Department/Division/WorkCenter code or name entered for filtering.

Therefore, it is absolutely CRITICAL that an agreed-upon code or name be established for each Division, and that code or name be CONSISTENTLY used by anyone involved in record maintenance or by those who have access to the database system.

These three data elements are no more than 10 alpha-numeric characters apiece, always in uppercase. If the names or codes don't exactly match, the filter may bypass a true record. Therefore, all users of the system need to ensure consistency.

However, the nature of this software system tends to have a low number of multiple users, so a consistent list of codes and names seems reasonable. The random selection process in the Urinalysis algorithm does not depend on or utilize these, or any of the other, data elements for list generation.

LIST OF REFERENCES

1. Burgess, Mark S., *Using Clarion Professional Developer*, First Edition, Addison-Wesley Publishing Company, 1991.

2. Date, C. J., *An Introduction into Database Systems*, Fourth Edition, Addison-Wesley Publishing Company, 1986.

3. Kroenke, David M., *Database Processing; Fundamentals / Design / Implementation*, Fourth Edition, Macmilliam Publishing Company, 1992.

4. Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, Third Edition, McGraw-Hill Inc., 1992.

5. Weitz, Lori, "4GLs keeping current with new technology; life being extended by support for GUIs, multiple DBMSs, rapid development", *Software Magazine*, May 1993 v13 n7, Sentry Publishing Company Inc., 1993.

6. Sommerville, Ian, *Software Engineering*, Fourth Edition, Addison-Wesley Publishing Company, 1992.

7. Rinehart, Martin L., "When to use an AppGen - and when not to.", *Data Based Advisor*, Oct 1993 v11 n10, Data Based Solutions Inc., 1993.

8. Chief of Naval Operations Instruction 5350.4 "Naval Alcohol and Substance Abuse Prevention Program", 1993.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93943-500	2
4.	Department of Management Sciences, Code AS Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor Norman F. Schneidewind Naval Postgraduate School Monterey, California 93943-5000	1
6.	William B. Short, CDR, SC, USN Naval Postgraduate School Monterey, California 93943-5000	1
7.	Dale E. Wilson, LT, USN Patrol Squadron 46 (VP-46) FPO AP 96601-5919	3